Adding Functionality to the HydraNet3 System

Santiago Salazar

CNT 4104 Software Project in Computer Networks

Dr. Janusz Zalewski

Florida Gulf Coast University

September 19, 2011

Table of Contents

Section 1: Introduction

Section 2: Definition of the Problem

Section 3: Prospective Solution

Section 4: Implementation

Section 5: Conclusion

Section 6: References

Appendix A: Use Manual

Appendix B: Server Source Code (C#)

Appendix C: HydraProp DLL Source Code (C#)

Appendix D: Main Web Site (ASP.NET)

Appendix E: Player Web Site (ASP.NET)

Appendix F: Player Client Applet Source Code (Java)

Appendix G: Developer Web Site (ASP.NET)

Appendix H: Developer Client Source Code (ASP.NET)

Appendix I: Hydra Game Example 1 Source Code (Spin)

Appendix J: Hydra Game Example 2 Source Code (Spin)

Appendix K: Hydra Game Example 3 Source Code (Spin)

Appendix L: Hydra Game Example 4 Source Code (Spin)

Section 1: Introduction

The HYDRA Game Development Kit is a platform used to learn game development. The HYDRA Game Development Kit was designed by Andre LaMothe [1]. He is the founder of Xtreme Games LLC, the Xtreme Games Developers Conference, and Nurve Networks LLC. This platform is based on a Parallax's propeller multiprocessor chip. This platform is known by its easy programmability, and thus game developers are only assumed to have some basic understanding of any programming language similar to C. The HYDRA Game Development Kit can be bought by $199.95 at www.xgamestation.com.

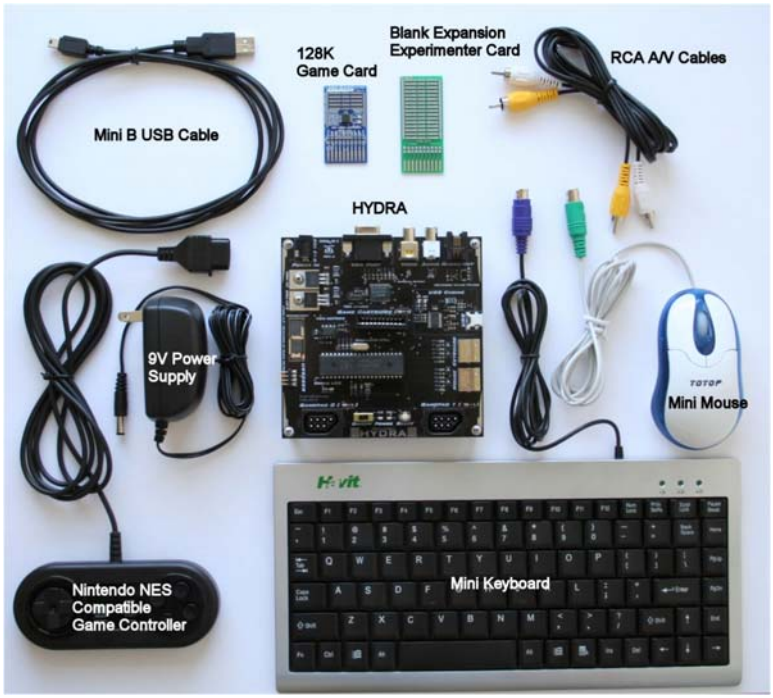

Figure 1.1 HYDRA Game Development Kit

1.1 HYDRA Game Development Kit Description

The HYDRA Game Development Kit, shown in figure 1.1, comes with everything that is needed in order to start developing games. It includes:

1. Hydra Game Console (HGC)
2. PS/2 Mouse
3. PS/2 Keyboard

4. 128 KB Game Cartridge
5. Blank Experimenter Card
6. Mini-USB cable
7. 9V power cord
8. RCA A/V cable
9. Nintendo-compatible controller
10. CD ROM

Next, a brief description of each part included in the HYDRA Game Development Kit is given.

Hydra Game Console (HGC): This is the main component of the kit. It is basically an electrical board along with the propeller chip. This is where the games run. See Figure 1.2.

PS/2 Mouse and PS/2 Keyboard: These two parts are basically a regular mouse and keyboard. They can be used to give input to the Hydra Game system.

128 KB Game Cartridge and Blank Experimenter Card: The game cartridge, or game card, can be placed in the expansion port of the Hydra console. The game card can be preloaded with games. Initially, it is preloaded with a game called “Ball Buster.” Thus, when the game card is inserted and the console is turned on, the Hydra will boot from the game card rather than the on-board serial EEPROM. If nothing is present in the game card, then the processor boots up from its main memory. The blank experimenter card allows to expand the functionality of the Hydra console, and works in a similar manner.

Mini-USB cable: This cable is used to connect the Hydra console to a computer.

9V power cord: This cable is used to connect the Hydra console to a power outlet.

RCA A/V cable: This cable is used to connect the Hydra console to a compatible TV.

Nintendo-compatible controller: These controllers are basically the same as the old NES controllers. They provide the player with a device to set input to the Hydra system.

CD ROM: This CD contains some developing tools such as the Propeller Tool used to develop programs for the Propeller chip.

Having discussed the HYDRA kit package contents, we proceed to briefly explain the HYDRA hardware.

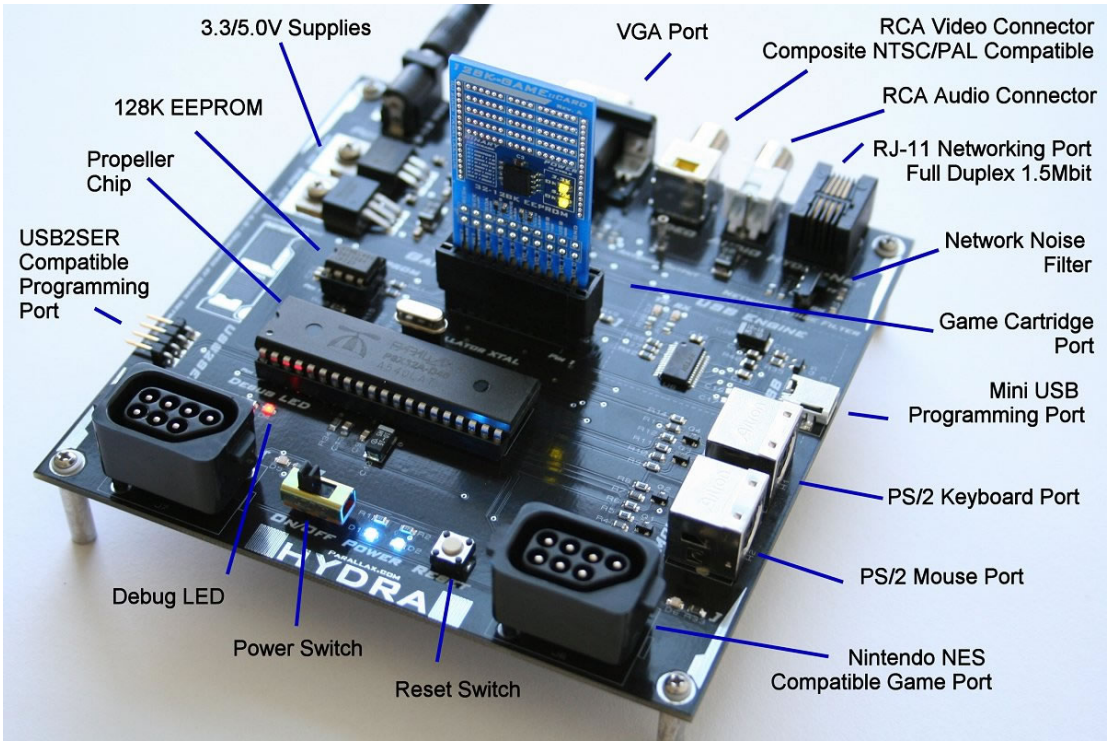

Figure 1.2: HYDRA Game console

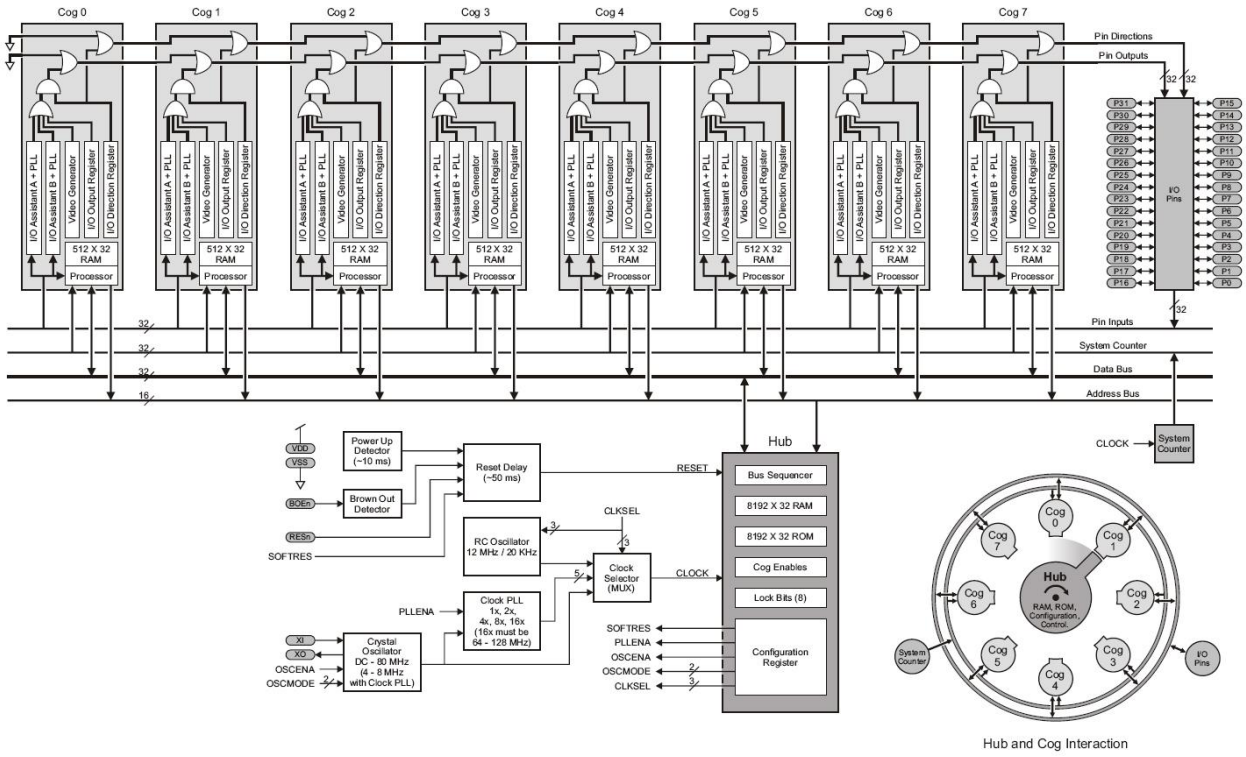

Figure 1.3 Propeller Chip Block Diagram

1.2 HYDRA Hardware Description

Propeller Chip: The HYDRA game console was built to work with the Parallax Propeller chip. The chip architecture is shown in Figure 1.3. The propeller chip was designed by Chip Gracey of Parallax Inc. The chip consists of eight cores, more commonly called cogs. These cogs are independent of each other. The chip is a 32-bit RISC-like architecture. Moreover, the propeller chip can run assembly language as well as a more high level language called Spin [3]. The chip executes only 32-bit instructions.

Main Memory: The amount of main memory provided in the HYDRA console is 64KB, which is organized in two blocks, each consisting of 32KB. The first block of memory is Random Access Memory (RAM), and the second block is Read Only Memory (ROM). The ROM contains an interpreter, certain data tables, and other objects needed for the proper operation of the Propeller chip. The RAM is used to load the user programs from the PC.

128 KB EEPROM: EEPROM stands for Electrically Erasable Programmable Read-Only Memory. Usually these chips store small amounts of information in a non-volatile way. That is, if the power is no longer supplied to the unit, the information or data is still saved in the unit. However, this technology provides the advantage of being erasable and programmable. Thus the data on such units could be updated as many times as needed, similar to a RW-CD.

Nintendo NES Compatible Game Ports: The Hydra console provides two Nintendo NES Compatible Game Ports. The controls connected to these ports send the input data to the game to be processed by the game console. The data received from the game controllers are parallel. The Hydra console also provides an LED that indicates whether or not the game controller is plugged into the system. A typical NES compatible control is shown in Figure 1.4.

Figure 1.4: NES Game Controller

Mini USB Programming Port: This port is the one that allows a connection between a PC and the Hydra console. A Mini-USB cable is provided in the Game Development Kit to realize this connection. Inconveniently, this cable is only about a foot long. However, this type of cable is common, and if needed, one can find longer Mini-USB cables at local stores.

1.3 HydraNet Overview

Having described the HYDRA Game Developer Kit, we next present an overview of the previous projects on the Hydra system. There are in existence three previous HYDRA projects. Thus before a description of the current project is given, a brief explanation of the accomplishments of the previous projects is summarized next.

At this point and due to the numerous extensions already made on the HydraNet project, it is difficult to have a concrete problem definition. Nevertheless, some ideas as presented next, but before we overview the previous projects: HydraNet 1, HydraNet 2, and HydraNet 3. HydraNet 1 is the first project using the HYDRA console. HydraNet 2 is extension of HydraNet 1. Finally, HydraNet 3 is yet another extension of HydraNet 2.

1.3.1 HydraNet 1

This is the first project on the HYDRA Game Development Kit developed by Scott Marks and Rick Haberkamp [2]. The goal of HydraNet 1 was to develop a game for two players. Each player would play on its own HYDRA console, and the communication between the HYDRA consoles would be implemented through a network, or the Internet. HydraNet1 consisted on two main components: The Hydra game and the computer program realizing the network interface through which the Hydra consoles would communicate.

The game was very simple in nature. This allowed avoiding complications since the main purpose of this project was to achieve a communication between the two Hydra consoles. It consisted of two sprites that moved around the screen, each controlled by one player. The purpose of the game, or player, is to obtain control over the enemy, namely, the other player. [1]

This project runs multiple threads at the same time. The game itself is a multithreaded program which utilizes the Propeller's multiple cores. Moreover, several communication threads were implemented on each Hydra console, one for receiving and another for sending game state information from and to the other Hydra console. As for the network interface of the system, which runs on the PC, four threads were required: one for reading the serial port, one sending to the serial port, one for reading incoming game state information, and finally another for sending game state information.

The setup for this multiplayer system is shown in Figure 1.4. The flow of data in this system is clearly depicted. A Hydra console sends its game video output to a TV. It also sends and receives game state information by communicating with the host PC. Also each host PC sends and receives each other's game state information through the network.

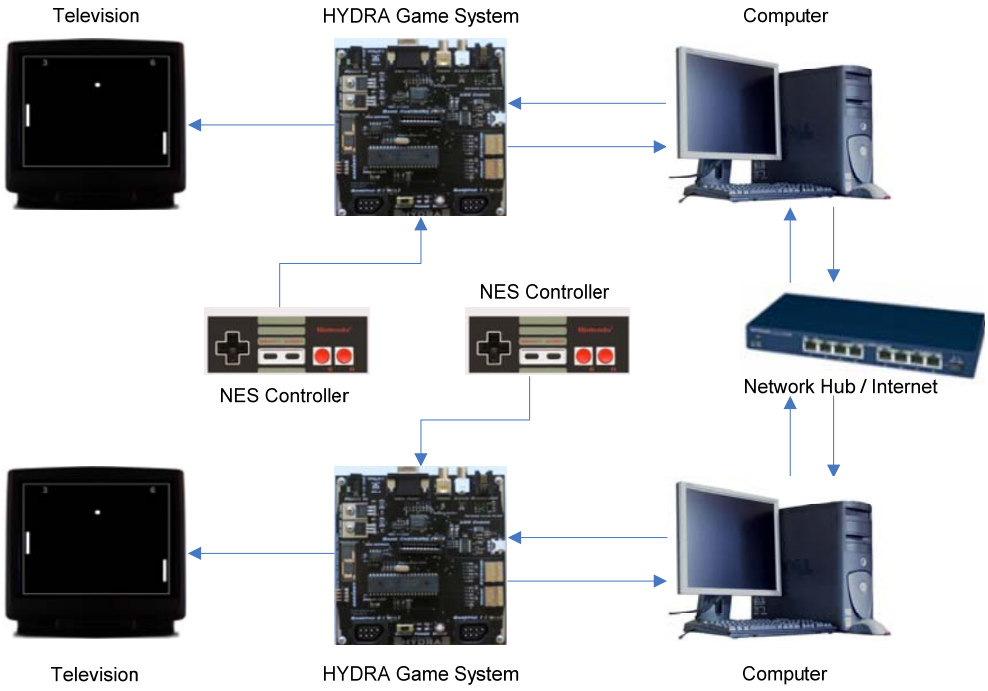

Figure 1.4: HydraNet1 Data Flow Diagram

In this architecture, each Hydra console is connected to an output screen. However, each of the consoles is also connected to a PC, which is in a network, or the Internet. Thus, the PC are able to communicate with each other and exchange information with the respective consoles. The flowchart of this architecture is depicted in Figure 1.5. it merely describes the network layer organization of the system. Each Hydra console assume the role of a sending and receiving clients simultaneously. In fact,

this is more similar to a peer-to-peer architecture. However, in code, this is implemented as a client-server architecture. A screenshot of the game is shown in Figure1.6.

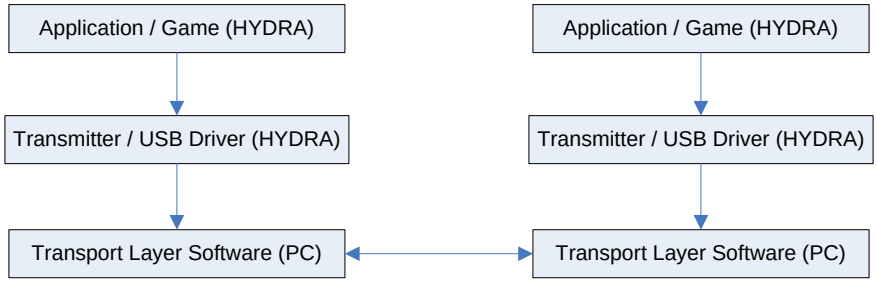
Application / Game (HYDRA)
Application / Game (HYDRA)
Transmitter / USB Driver (HYDRA)
Transmitter / USB Driver (HYDRA)
Transport Layer Software (PC)
Transport Layer Software (PC)

Figure 1.5: Flowchart of HydraNet1 System

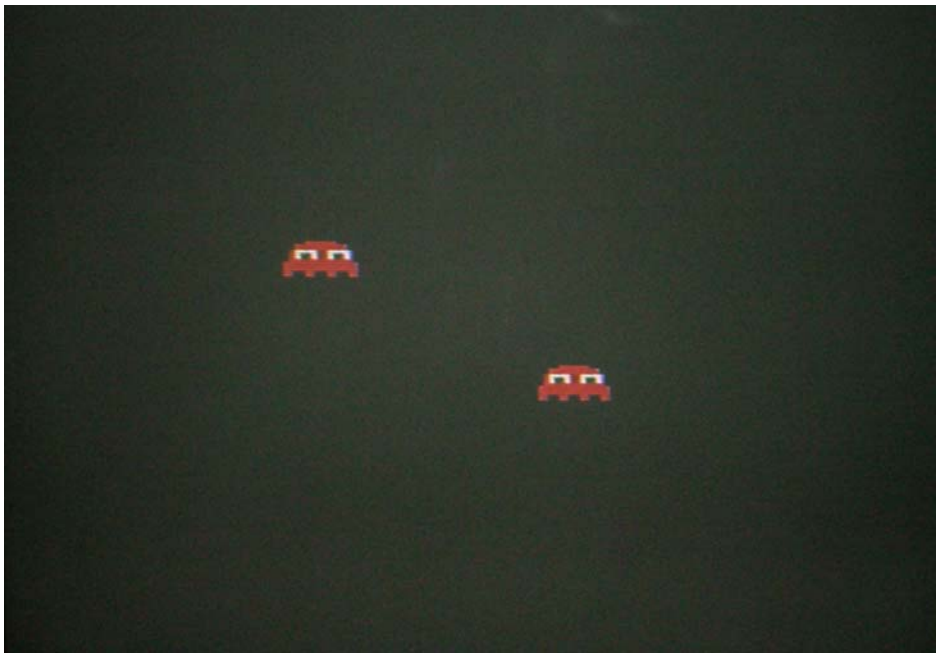

Figure 1.6: Game ScreenShot

1.3.2 HydraNet 2

This is the second HYDRA project developed again by Scott Marks and Rick Haberkamp. The purpose of this project was to extend the HydraNet 2 to allow multiplayer capabilities. HydraNet 2 also incorporates a server-client architecture. Each player, or client, would connect to a server over the internet. [1]

A sample screenshot of the server and client running in this system is shown in Figure 1.7. As one can see in the picture, the server (top part) displays an IP address, and chooses a port number to which it listens for clients trying to connect. The server runs in an infinite while loop, waiting for players to connect. On the other hand, in the client side, the users inputs the IP address and the port number and hits enter. Then the connection is established. The architecture of this system is shown in Figure 1.8. As one can see, several clients can connect to the server PC to which the Hydra console is connected. Thus, this server runs as an interface between the players on the network and the Hydra console. Also, the network layer organization for this system is shown in Figure 1.9.

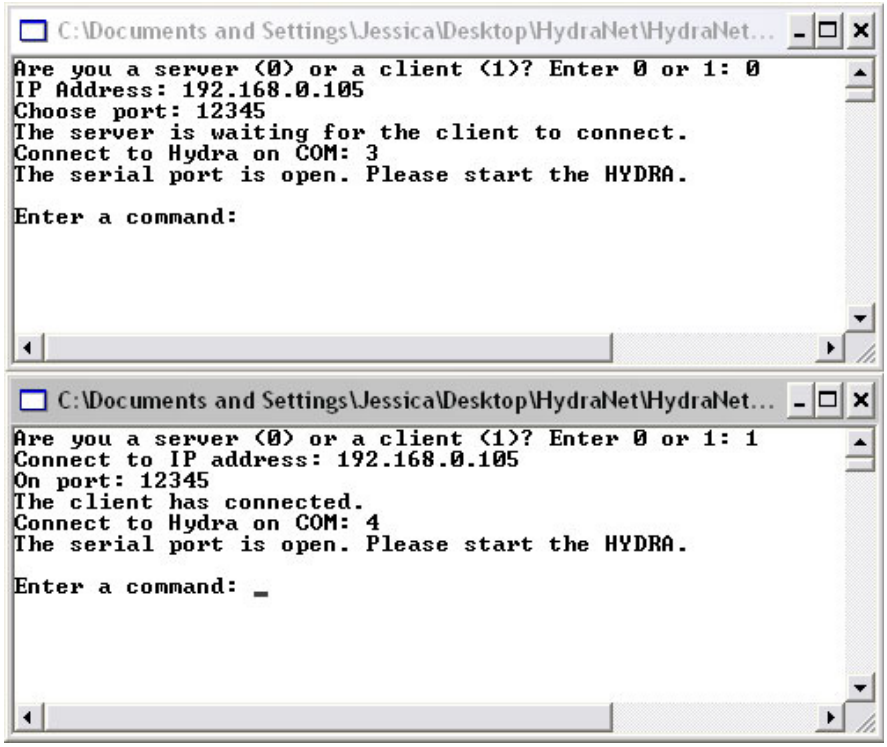
C:\Documents and Settings\Jessica\Desktop\HydraNet\HydraNet...
Are you a server (0) or a client (1)? Enter 0 or 1: 0
IP Address: 192.168.0.105
Choose port: 12345
The server is waiting for the client to connect.
Connect to Hydra on COM: 3
The serial port is open. Please start the HYDRA.
Enter a command:
C:\Documents and Settings\Jessica\Desktop\HydraNet\HydraNet...
Are you a server (0) or a client (1)? Enter 0 or 1: 1
Connect to IP address: 192.168.0.105
On port: 12345
The client has connected.
Connect to Hydra on COM: 4
The serial port is open. Please start the HYDRA.
Enter a command:

Figure 1.7: HydraNet Server (top) and Client (bottom)

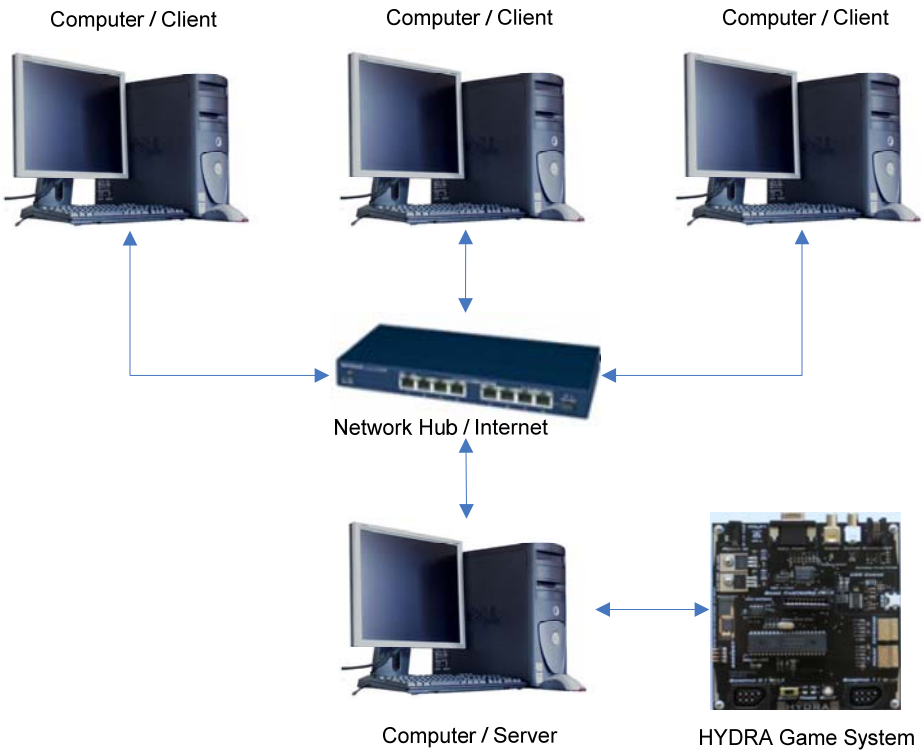

Figure 1.8: HydraNet2 Architecture

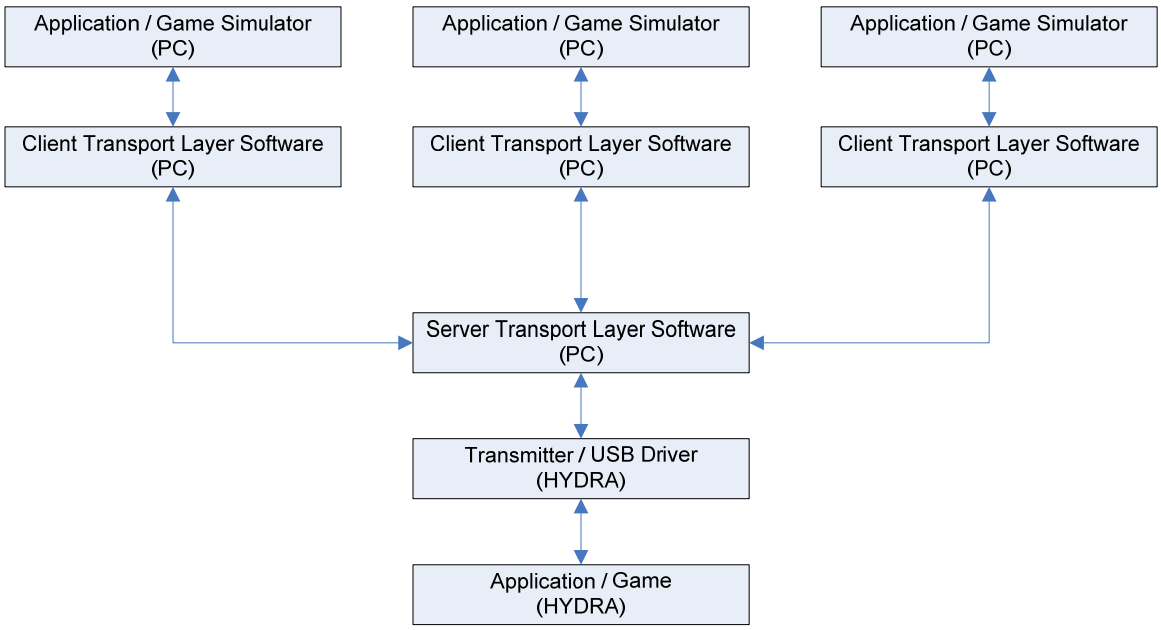
Application / Game Simulator (PC)
Application / Game Simulator (PC)
Application / Game Simulator (PC)
Client Transport Layer Software (PC)
Client Transport Layer Software (PC)
Client Transport Layer Software (PC)
Server Transport Layer Software (PC)
Transmitter / USB Driver (HYDRA)
Application / Game (HYDRA)

Figure 1.9: Basic Network Layer Organization

1.3.3 HydraNet 3

This, being the third HYDRA project, is yet again another extension of HydraNet 2. However, this project was solely created by Scott Marks. In this extension, game graphics and a web interface are introduced. It also allows developers to create games for the HydraNet system. [1]. The main website consists of a player web site and a developer website. The players can play with the Hydra console through the player's web site. However, if one want to develop game for the Hydra console, which are written in Spin language. The developer's web site provides instruction on how to do so. Thus the users, or better yet, developers, can create their own games, uploaded into the Hydra console thorough the server, and then start playing their own games.

The instructions of how to upload games into the Hydra console from the Internet are as well provided on the developer's web site. The user just basically needs to upload a set of three files into the system. The first, and most important one, being the executable file of the actual game to program. The second one being the source code, and possibly a zip file containing pictures if they are used in the game. Finally, the system requires a text file named manual to be uploaded as well. This is basically a user's manual that describes how to play the game. When this set of files are uploaded successfully through the developer's page, a message saying "success" is displayed back to the screen.

1.3.4 HydraNet 4

This project is the last extension of the Hydra system. It is developed by Jinlong Fan. In this instance, the contents of the Hydra game re enriched, the characters can now move their heads and legs, and the number of pictures is increased, thus augmenting the graphical aspects of the game. This project dramatically improves the graphics of the older game, which is implemented in the following way: First, the computation of coordinates is computed by the Hydra console as the player's input is received. Second, these coordinates are sent back to the server hosting the Hydra console, and the Java applet takes these coordinates and uses them to display pictures on the screen. Thus the end user is able to see them in their web browser.

Section 2: Definition of the Problem

Currently, the HydraNet3 system supports online multiplayer gaming. However, the game itself is very simple in nature, since the goal of this project was to achieve multiplayer capabilities through a client-server architecture. In this system, the Hydra console is connected to a server computer. Client computers can connect to the HydraNet3 system through the following website: http://69.88.163.18/samarks/Default.aspx. Detailed information about the previous projects can be found here as well. In order to play with the HydraNet3 system, the server needs to run a program that constantly checks for incoming connection. The server is set up the server in a way that this program is constantly running. Then, any computer connected to the Internet can visit the HydraNet3 website and go to the HydraNet3 player link. A Java applet will open up, and the game is contained there. Then the user can start playing along with other connected players.

One of the most feasible extensions to the HydraNet3 system is to add more games. Respective links to the new games can be placed in the HydraNet3 system's web site. Thus, players have the opportunity to choose from a variety of games when connecting to the HydraNet3 system. However, one has to keep in mind that the Hydra console can run only one game at any given moment.

In order to play, a user needs to go to the following website: http://69.88.163.18-/samarks/-Default-.aspx as shown in the Figure 2.1. Then click on the link: "Player page". This would take you to the website shown in Figure 2.2. Then click anywhere inside the Java Applet, and the game will start. The game controls to move the players are as follows:

- A: Move Left
- D: Move Right
- W: Move Up
- S: Move Down
- J: Action/Turbo

With these controls, each user can move their own player and have interaction with other currently connected players.

HydraNet 3 Gaming System

CEN 4935 Senior Software Engineering Project (Spring 2009)
By Scott Marks

The purpose of this project is to expand and improve HydraNet 2 in two ways: the addition of game graph
HydraNet system without a significant reduction in performance. The web interface, with the game embedd

View the entire HydraNet 3 project report entitled "HYDRA Network Control and Graphics."

HydraNet 3 Player

The Player page can be used to play the current game.

View the game manual (opens in new window/tab).

The game controls are:

- A: Left
- D: Right
- W: Up
- S: Down
- J: Action

HydraNet 3 Developer

Figure 2.1: Hydra Main Web Site

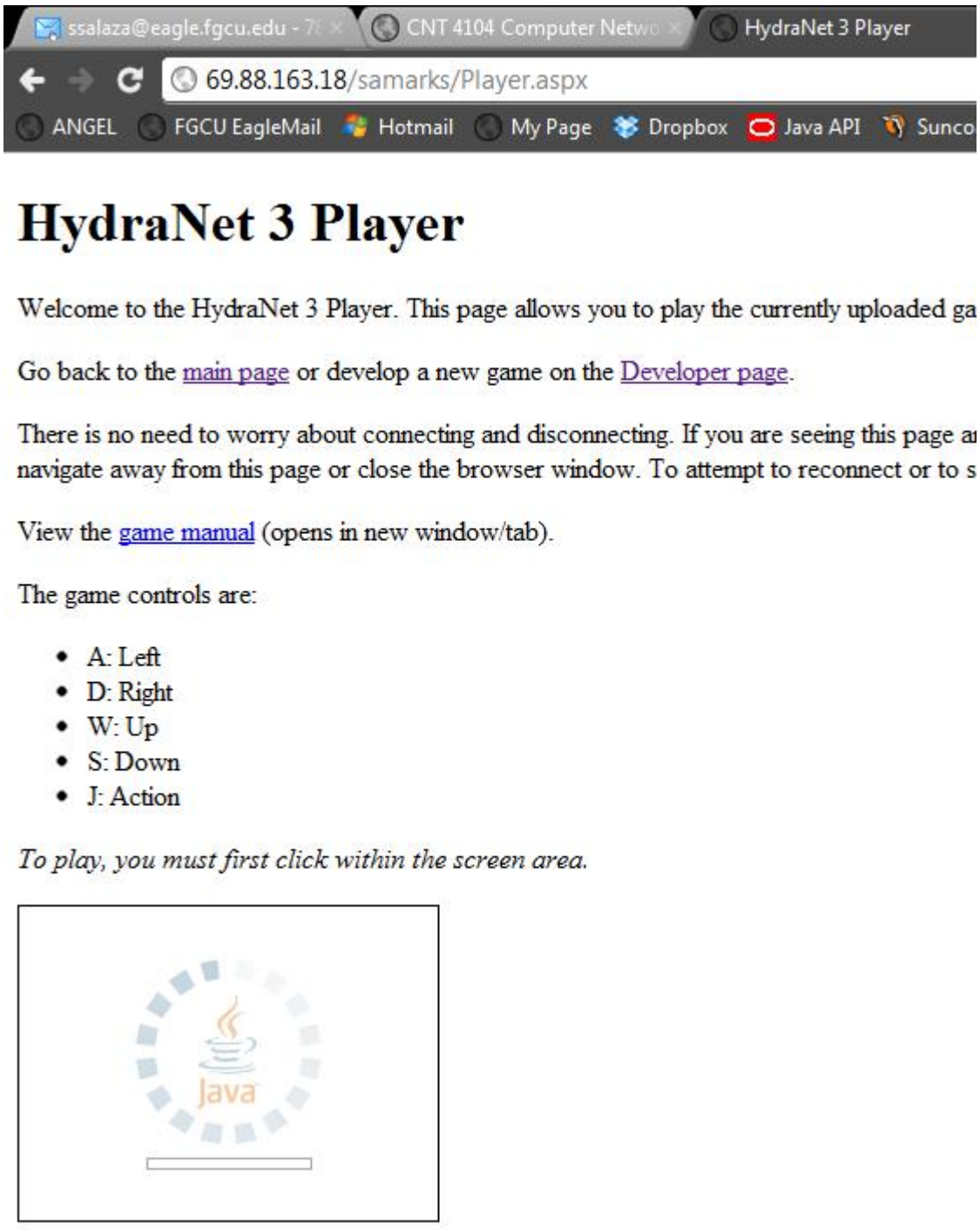
ssalaza@eagle.fgcu.edu - 7
CNT 4104 Computer Netwo
HydraNet 3 Player
69.88.163.18/samarks/Player.aspx
ANGEL
FGCU EagleMail
Hotmail
My Page
Dropbox
Java API
Sunco
HydraNet 3 Player
Welcome to the HydraNet 3 Player. This page allows you to play the currently uploaded ga
Go back to the main page or develop a new game on the Developer page.
There is no need to worry about connecting and disconnecting. If you are seeing this page a
navigate away from this page or close the browser window. To attempt to reconnect or to s
View the game manual (opens in new window/tab).
The game controls are:
A: Left
D: Right
W: Up
S: Down
J: Action
To play, you must first click within the screen area.
Java

Figure 2.2: Hydra Player Web Site

Section 3: Prospective Solution

Adding more games to the HydraNet3 system consists of building up games in the Spin language, and making them available to players that connect to the system. However, this is not an easy task.

Before adding a game, we need to code it. Thankfully, one can find many classic games available along with their respective source code from the documentation provided with the Hydra Game Development Kit. Thus for instance, one of these games could be chosen. The source code of the chosen game would be slightly modified depending on the needs of the HydraNet4 system. A directory containing all the added games would be added to the HydraNet3 system. Then the compiled files and the source code for each game would be placed in the newly created games directory. The website would be slightly modified. First it will provide links to the newly added games, so the client player would have the ability to choose from a variety of games. Once a player loads a game, the game will start to run in the Hydra console.

The games would have to be modified in such a way that the input always comes from the client players computers, which are of course in remote locations. Moreover, the output of the Hydra console would have to be redirected at each client's computer through the Java applet.

Since for each game, we need four files, the respective links are added to the main web site as shown in Figure 3.1. The first file is the compiled executable file with the .eeprom extension. The second file is the source code in .txt format. The third file is just a dummy .zip file. Finally, the fourth file is a user manual in a .txt extension.

- Example 1 shows how to get 4 players to move about a static background.
  - Compiled game
  - Source code
  - Images (zipped)
  - Manual
- Example 2 shows how to get 4 players to move between two backgrounds and how to get the player images to change based on movement and turbo state.
  - Compiled game
  - Source code
  - Images (zipped)
  - Manual
- Example 3 is a sample program that turns ON/OFF the debug led in the HYDRA console
  - Compiled game
  - Source code
  - Images (zipped)
  - Manual
- Example 4 is an classical game
  - Compiled game
  - Source code
  - Images (zipped)
  - Manual

Figure 3.1: Links to New Games

Section 4: Implementation

The HydraNet3 system allows developers to develop and upload games to the Hydra console via its web site. Instructions of how to do so can be found there as well. Thus a mechanism for loading games to the Hydra console from a remote location is already in existence.

This mechanism is slightly modified in such a way, that after the client players chose a game to play, a sequence of instructions is generated to load the game into the Hydra console. After loading the appropriate game in the Hydra console, the client player can proceed to play the game.

Some Spin programs have been successfully loaded into the Hydra console. But only through the developer tool that comes with the kit. Therefore, at least theoretically, this is possible.

Some games and Spin programs have been successfully uploaded to the HydraNet web site as shown in Figure 4.1. The respective links have been placed there as well. The new games are basically Example 3 and Example 4. Example 3 is a spin language program that turns ON/OFF the debug led in the Hydra console. Note that in the source code, the frequency can be changed as shown in Figure 4.1. This frequency determines the speed at which the debug LED turn ON and OFF.

For Example 3, a code segment in which the user can change the frequency of the clock ticks is shown in Figure 4.1.

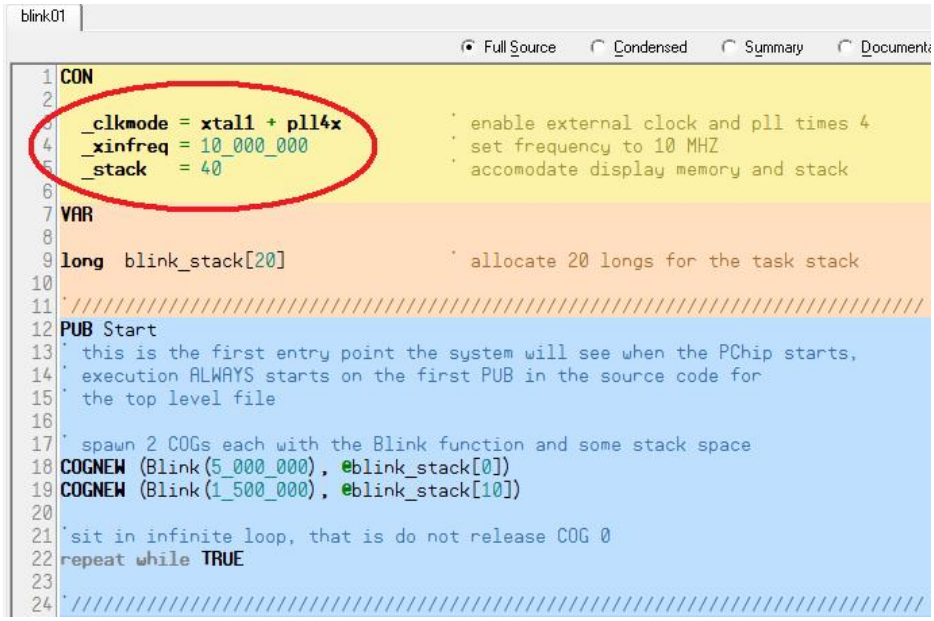
blink01
Full Source
Condensed
Summary
CON
_clkmode = xtal1 + pll4x
_xinfreq = 10_000_000
_stack = 40
enable external clock and pll times 4
set frequency to 10 MHZ
accomodate display memory and stack
VAR
long blink_stack[20]
allocate 20 longs for the task stack
PUB Start
this is the first entry point the system will see when the PChip starts,
execution ALWAYS starts on the first PUB in the source code for
the top level file
spawn 2 COGs each with the Blink function and some stack space
COGNEW (Blink(5_000_000), @blink_stack[0])
COGNEW (Blink(1_500_000), @blink_stack[10])
sit in infinite loop, that is do not release COG 0
repeat while TRUE

Figure 4.1: Code segment a SPIN program

Example 4 is a classical game called Mars Landers. In this game, each player tries to control a spaceship, in which fuel is limited. The purpose of this game is to try to land in the respective area before the ship runs out of fuel. This game can be uploaded to the Hydra console from the network but not played through the Java applet. However, it can still be played in the Hydra console, provided that its output is connected to a TV through RCA cables.

Section 5: Conclusion

To conclude, several more games have been added to the HydraNet system. Respectively, the source code, the user manual, the dummy zip files, and the executables have been included in a newly created folder residing on the server PC. The respective links for downloading these files have been added to the website as well as shown in Figure 5.1. The reason why a dummy zip file is provided is because the HydraNet system requires a .zip file when the developer is trying to upload a game into the Hydra console. Since the new games do not use pictures as a way of generating graphics, the .zip dummy files are available for download.

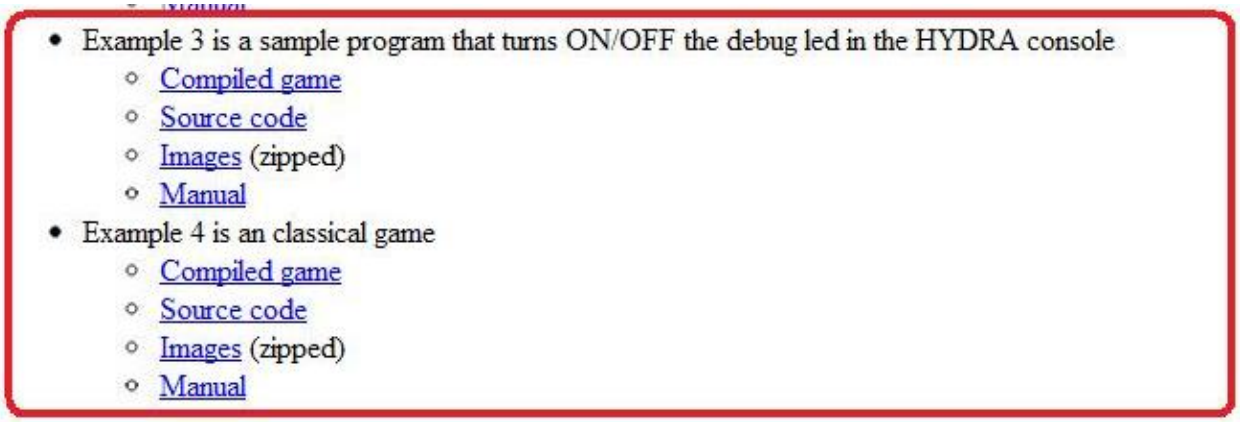

Figure 5.1: Respective links to new games

For testing purposes, a special program was created. This program is located under Example 3. The purpose of this program is to turn ON/OFF the debug LED in the Hydra console. With this program, one can test whether or not games can be successfully loaded into the Hydra console. For instance, if we load this program successfully, the debug LED of the Hydra console should immediately start blinking at the frequency set in the source code. Loading this program has been done on several occasions and from different computers on different networks. This implies that all the games loaded in this manner are successfully uploaded into the Hydra console.

In any case, Example 4, which is a real game (in fact, it is the classical game Mars Landers!) have been successfully loaded several times. Nevertheless, this game does not use pictures to generate game graphics and thus the output is not sent back to the Java applet. Rather, the output is sent to the regular video output of the Hydra console.

Since a TV monitor is not available in the lab, an attempt was made, using several methods, to display the output of the Hydra console on various Pc monitors. For this reason, a RCA to VGA converter was purchased. Unfortunately, this did not work. However, the Hydra console was connected to several TV's after successfully uploading Mars Landers. As a future extension, more games can be added to the HydraNet system in a similar way.

Several attempts were made to make the newly added games playable online. However, this was impossible due to the high complexity of the drivers that would be required to send the graphics generated in the Hydra console back to the server. For instance, most of these games use set of drivers to send the generated graphics to the regular RCA video output. These drivers are written in the native assembly language of the Propeller chip, and each of them is several thousands of lines of code. Although, the newly added games are not available to play online, they can be locally played on the Hydra console using the NES compatible controller.

One final aspect that is worth mentioning is that the complete source code including all the previous reports have been reorganized, along with the new additions of this project, and placed in the C:/ samarks/ directory on the 69.88.163.18 server.

Data Analysis

Data analysis is an important aspect of this project because data is flowing all over the place in the Hydra System. For instance consider a game developer uploading a game. In this situation, the game developer has to upload a few files including the source code of the spin program. First, the data flows from the end user through a network, which then arrives at the server. Then the server takes over, and uploads the data directly into the Hydra console. Thus, it is important to make sure that the data flowing at any particular time during the operation of the Hydra system is consistent. To address this issue, the Hydra system uses a variety of methods.

One particular and important method is that when a game developer is going to upload any game into the Hydra console, the Hydra system checks the validity of these files by checking that they are in the right format. For example, the system checks that the source code of the game is .spin. If the format of the files does not coincide with the specifications and requirements, then loading such files will not

be successful. Notice that in this manner, we can prevent to a certain degree malicious files such as worms and viruses being uploaded to the server, and consequently to the hydra console.

Another particular method used by the Hydra system is that any game to be uploaded has to be in a compiled format. In other words, besides uploading the source code of a game, it is also required to upload an object format of the game. This is actually the file that will be uploaded to the Hydra console. Since this file has been already compiled by the Propeller Tool, it is guaranteed that the server will successfully be able to upload that particular file into the Hydra console.

Finally, another method that tries to address the issue of safety in regards to the flow of data is that for the most part, the developers of the Hydra System have total control of the data once it has been uploaded to the computer server hosting the Hydra console. Although this method is less significant than the others, it is important to note that no people connected to the Hydra system can in any way have control of the data flow, except when playing a video game, which is very secure and stable since there are only a predetermined set of inputs, and a game has specific handlers to take care of those inputs.

Section 6: References

[1] “HYDRA Game Development Kit.” XGameStation: Video Game System Development. 2011. Nurve Networks, LLC. 12 October 2011. < http://www.xgamestation.com/view_product.php?id=33 >

[2]Marks, S. and Haberkamp, R. “Programming an Interface for the HYDRA game Development Kit.” April 28, 2008. “HYDRA Network Control and Visualization.” December 15, 2008. Marks, S. “HYDRA Network Control and Graphics.” March 11, 2009. < http://69.88.163.18/samarks/Default.aspx>. Florida Gulf Coast University.

[3] LaMothe, A. Game Programming for the Propeller Powered HYDRA. 2006. Nurve Networks, LLC.

[4] “Propellent Library.” Propellent (Library and Executable). 2008. Parallax, Inc. 8 Oct 2008. <http://www.parallax.com/tabid/442/Default.aspx>.

[5] “XGameStation Pico Edition Kit 2.0.” XGameStation: Video Game System Development. 2008. Nurve Networks, LLC. 13 Oct 2008. <http://www.xgamestation.com/view_product.php?id=34>.

[6] Fan, J. “Hydra Network Control and Graphics.” April 30, 2010. Florida Gulf Coast University.

Appendix A: User Manual

First, we describe the steps to set up the server so that the Hydra server runs at system startup. In this manner, there does not have to be a user logged in order to run the Hydra server. Rather the computer does this in the background. These steps are described next:

1. Open notepad and write the following line: C:/Inetpub/wwwroot/smarks/HydraServer.exe
2. Save this file as a .bat file
3. Place this file on the following directory: C:\Windows\System32\GroupPolicy\Machine\Scripts\Startup
4. Click on start>run and type in “gpedit.msc”
5. On the window that opens up. Go to Windows Settings
6. Double click on Scripts(Startup/Shutdown)
7. Double click on Startup
8. Click on Add…
9. Click on Browse…
10. Now go to the directory where you have previously saved the .bat file and select it
11. Click on open, and click ok two consecutive times

Now, every time the server is turned on, the HydraServer.exe is executed on the background without the need of any user logging in.

Now, the following are the set of steps to set up the Hydra console and start uploading other games as well as playing. To do this, apply the following procedure:

1. Connect the Hydra console to the host PC through the USB cable
2. In the host PC, go to C:/Inetpub/wwwroot/smarks

3. Double click on HydraServer.exe. Now the Hydra server should be running and listening for connections. A screenshot is presented in Figure A.1.

HydraNet 3 Server
Welcome to the HydraNet 3 Server.
Connecting to the HYDRA...
Could not connect to HYDRA. Try again? [y/n]: y
The server is connected to the HYDRA.
Server is running...
Type "help" for a list of commands.

Figure A.1: HydraNet server running

4. The user can type “help” and the hit enter on the command window, and some available commands will be displayed as shown in Figure A.2.

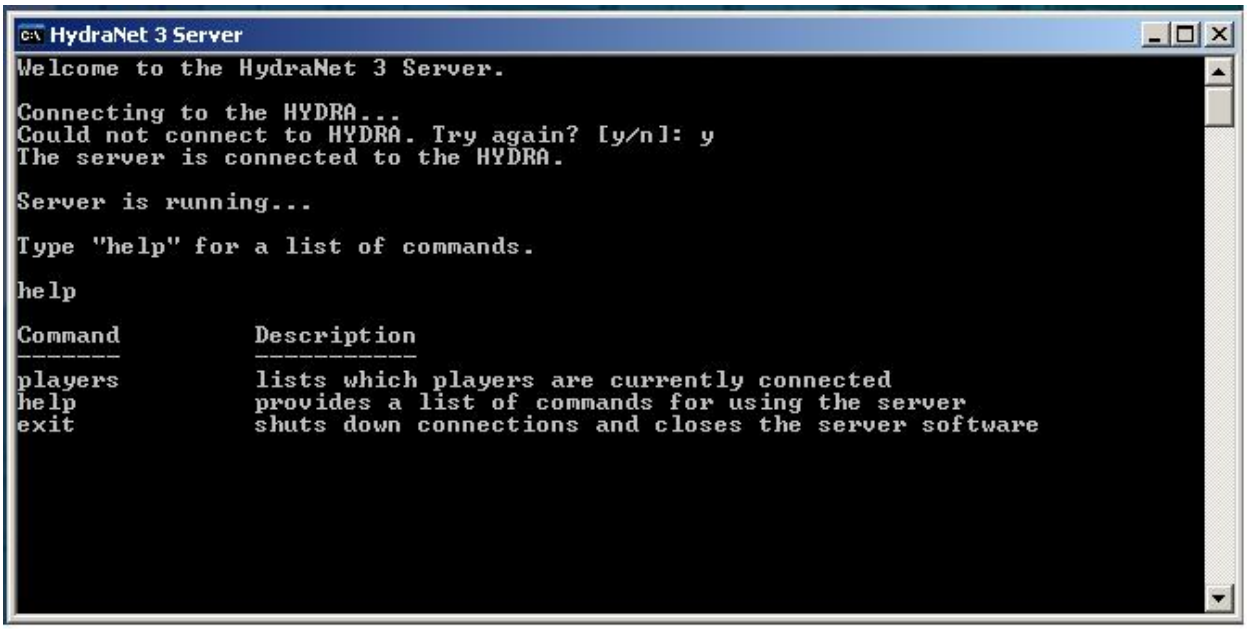
HydraNet 3 Server
Welcome to the HydraNet 3 Server.
Connecting to the HYDRA...
Could not connect to HYDRA. Try again? [y/n]: y
The server is connected to the HYDRA.
Server is running...
Type "help" for a list of commands.
help
Command Description
players lists which players are currently connected
help provides a list of commands for using the server
exit shuts down connections and closes the server software

Figure A.2: HydraNet server help command

5. The user can then type players and hit enter to see a list of all currently connected players as shown in the Figure A.3.

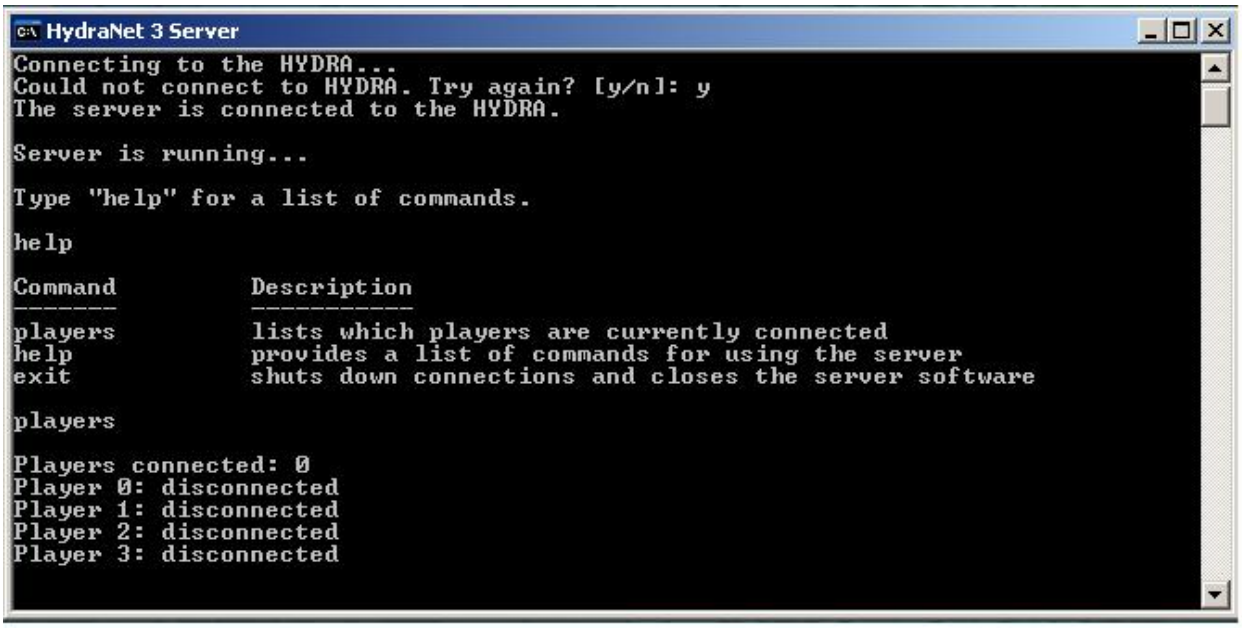
HydraNet 3 Server
Connecting to the HYDRA...
Could not connect to HYDRA. Try again? [y/n]: y
The server is connected to the HYDRA.
Server is running...
Type "help" for a list of commands.
help
Command Description
players lists which players are currently connected
help provides a list of commands for using the server
exit shuts down connections and closes the server software
players
Players connected: 0
Player 0: disconnected
Player 1: disconnected
Player 2: disconnected
Player 3: disconnected

Figure A.3: HydraNet server players command

6. To close the server, just type “exit” and then hit enter

7. After the server is running and listening to connections, go to the following website: http://69.88.163.18/samarks/Default.aspx.

8. To play the currently loaded game in the Hydra console, click on the player page link, which would take you to the following web site: http://69.88.163.18/samarks/Player.aspx.

9. Once in this website click inside the Java applet. Then the game will start as shown in Figure A.4.

View the game manual (opens in new window/tab).

The game controls are:

- A: Left
- D: Right
- W: Up
- S: Down
- J: Action

To play, you must first click within the screen area

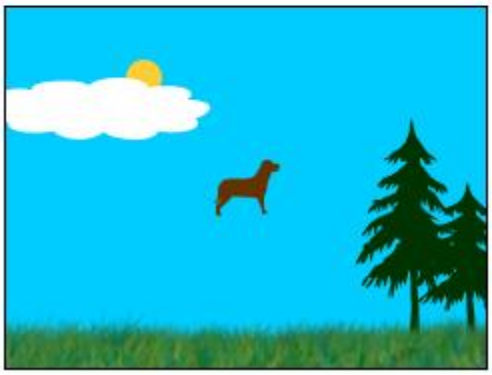

Figure A.4: Java applet containing a game

Appendix B: Server Source Code (C#)

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using System.IO;
using System.IO.Ports;
using System.Threading;
using System.Reflection;
using HydraProp;

namespace HydraServer
{
    class HydraServer
    {
        #region Variables and classes
        // a simple class to store player info
        class Player
        {
            public Thread thread;
            public TcpClient conn;
            public int playerNumber = 0;
            public bool active = false;
        }

        // maximum allowed players
        const int MAX_PLAYERS = 4;

        // variables to keep track of connected players
        int numPlayers = 0;

        // variables for connections
        TcpListener server;
        Player[] players = new Player[MAX_PLAYERS];
        TcpClient devConn;
        SerialPort hydra;

        // variables for other threads
        Thread consoleThread, hydraThread;

        // command codes for players; values must be in range of 241-254
        const byte PLAYER0_CONNECT = 241,
            PLAYER1_CONNECT = 242,
            PLAYER2_CONNECT = 243,
            PLAYER3_CONNECT = 244,

            PLAYER0_DISCONNECT = 245,
            PLAYER1_DISCONNECT = 246,
            PLAYER2_DISCONNECT = 247,
            PLAYER3_DISCONNECT = 248,
```

```
    DEV_UPLOADING = 251,
    DEV_FINISHED = 252,

    SERVER_SHUTDOWN = 254,

    PLAYER_INPUT_OFF = 0;  // player's off signal
#endregion

HydraServer()
{
    // title and welcome message
    Console.Title = "HydraNet 3 Server";
    Console.WriteLine("Welcome to the HydraNet 3 Server.");

    // connect to the HYDRA
    connectToHydra();

    // start the server
    // gets the computer's IP address and the port number is 5309
    server = new TcpListener(IPAddress.Any, 5309);
    server.Start();
    Console.WriteLine("Server is running...\n");

    // create the player objects
    for (int i = 0; i < MAX_PLAYERS; i++)
    {
        players[i] = new Player();
        players[i].playerNumber = i;
    }

    // start the command console in a new thread
    consoleThread = new Thread(commandConsole);
    consoleThread.Start();

    // start HYDRA communications in a new thread
    hydraThread = new Thread(hydraToPlayers);
    hydraThread.Start();

    // variables to accept new clients
    TcpClient conn;
    Stream connStrm;

    while (true)
    {
        // accept a connection
        conn = server.AcceptTcpClient();
        connStrm = conn.GetStream();

        try
        {
            int clientType = connStrm.ReadByte();
            if (clientType == 1 && numPlayers < MAX_PLAYERS)
            {
                // server was not full, accept connection
                connStrm.WriteByte(1);
```

```
                // send client the number of images it needs to
            download
                connStrm.WriteByte((byte)getNumGraphics());

                // receive all-clear from client that downloading is
            complete
                if (connStrm.ReadByte() == 1)
                {
                    for (int i = 0; i < MAX_PLAYERS; i++)
                    {
                        if (!players[i].active)
                        {
                            // setup player
                            players[i].conn = conn;
                            players[i].active = true;

                            // start player thread
                            players[i].thread = new
            Thread(playerToHydra);
                            players[i].thread.Start(players[i]);

                            // end search for vacant player slot
                            break;
                        }
                    }
                }
            }
            else if (clientType == 0)
            {
                // save the connection
                devConn = conn;

                // handle the developer's uploaded files
                developerUpload();
            }
            else
            {
                // server is full, deny connection
                connStrm.WriteByte(0);

                connStrm.Close();
                conn.Close();
            }
        }
        catch (Exception)
        {
            Console.WriteLine("Error connecting client.");
            break;
        }
    }
}

#region Client communications
// handles communication with player clients
void playerToHydra(object p)
{
    numPlayers++;  // increment the number of connected players
```

```
Player player = (Player)p;

// get TcpClient object from parameter
TcpClient client = player.conn;
Console.WriteLine("Player " + (player.playerNumber + 1) + "
      connected.");

// get stream from player client
Stream connIn = client.GetStream();

// send message to all active players that a new player has
      joined
broadcastToPlayers((byte)(PLAYER0_CONNECT +
      player.playerNumber));

byte[] outByte = new byte[1];
byte inByte,  // byte to read player inputs from client
    pNum = (byte)player.playerNumber;  // save player number as
      byte
while (true)
{
    try
    {
        // if the stream is closed, stop reading from client
        if ((inByte = (byte)connIn.ReadByte()) < 0 || inByte
      >=255)
        {
            break;
        }
        else
        {
            // make the byte by including input and player number
            outByte[0] = (byte)(inByte | pNum);

            // send the input byte to the HYDRA
            sendToHydra(outByte);
        }
    }
    catch (Exception)
    {
        // end the loop for receiving game inputs
        break;
    }
}

// close the connection with the player client
try
{
    client.Client.Close();
    client.Close();
}
catch (Exception)
{
    // do nothing
}
```

```
    // send player off input to HYDRA just in case the player did not
          actually
    // transmit it before the connection was terminated
    outByte[0] = (byte)(PLAYER_INPUT_OFF | pNum);
    sendToHydra(outByte);

    Console.WriteLine("Player " + (player.playerNumber + 1) + "
          disconnected.");

    numPlayers--;  // decrement the number of connected players

    player.conn = null;
    player.active = false;  // deactivate player

    // send out player disconnect message
    broadcastToPlayers((byte)(PLAYER0_DISCONNECT +
          player.playerNumber));
}

// receives game state from HYDRA and sends to player clients
void hydraToPlayers()
{
    byte rec;
    while (true)
    {
        try
        {
            rec = (byte)hydra.ReadByte();

            // send game state to all active players
            broadcastToPlayers(rec);
        }
        catch (Exception)
        {
            // do nothing
        }
    }
}

// sends a byte message to all active players
[MethodImpl(MethodImplOptions.Synchronized)]
void broadcastToPlayers(byte message)
{
    // send the byte to all active players
    for (int i = 0; i < MAX_PLAYERS; i++)
    {
        if (players[i]!=null && players[i].active)
        {
            try
            {
                players[i].conn.GetStream().WriteByte(message);
            }
            catch (Exception)
            {
                // do nothing
            }
        }
```

```
    }
}

// sends a byte message to the HYDRA
[MethodImpl(MethodImplOptions.Synchronized)]
void sendToHydra(byte[] message)
{
    try
    {
        hydra.Write(message, 0, 1);
    }
    catch (Exception)
    {
        // do nothing
    }
}

void developerUpload()
{
    Console.WriteLine();
    Console.WriteLine("Developer uploading new game...");

    // get the developer connection stream
    Stream connStrm = devConn.GetStream();

    // recieve notification that developer is starting to upload
    if (connStrm.ReadByte() != 1)
    {
        Console.WriteLine("Connection with developer failed.");
        return;
    }

    // shut down the player threads and tell players that a new game
          is being uploaded
    for (int i = 0; i < MAX_PLAYERS; i++)
    {
        if (players[i].active)
        {
            players[i].thread.Abort();
            players[i].thread = null;
            players[i].conn.GetStream().WriteByte(DEV_UPLOADING);
        }
    }

    // shut down HYDRA thread and serial connection
    hydraThread.Suspend();
    hydra.Close();

    // send notification that other connections are stopped
    connStrm.WriteByte(2);

    // recieve notification that developer has finished saving
          uploaded files
    if (connStrm.ReadByte() != 3)
    {
        Console.WriteLine("Connection with developer failed.");
        return;
```

```
    }

    // get the file path
    String codePath =
          Assembly.GetExecutingAssembly().GetName().CodeBase;
    String filePath = Path.GetDirectoryName(codePath).Substring(6) +
        "\\game_files\\game.eeprom";

    // run Propellent for downloaded file
    Console.Write("Uploading game to HYDRA... ");
    HydraPropForm.uploadGame(filePath);
    Console.WriteLine("done.");

    // send notification that HYDRA upload is complete
    connStrm.WriteByte(4);

    // close connection with developer
    connStrm.Close();
    devConn.Close();

    // restart HYDRA serial connection and thread
    hydra.Open();
    hydraThread.Resume();

    // shut down all player connections
    Console.Write("Disconnecting players... ");
    for (int i = 0; i < MAX_PLAYERS; i++)
    {
        if (players[i].active)
        {
            players[i].conn.GetStream().WriteByte(DEV_FINISHED);
            players[i].conn.Close();
            players[i].conn = null;
            players[i].active = false;
        }
    }
    numPlayers = 0;
    Console.WriteLine("done.");

    Console.WriteLine("Game upload complete.");
    Console.WriteLine();
}
#endregion

#region Miscellaneous methods
// checks for commands from the server console
void commandConsole()
{
    Console.WriteLine("Type \"help\" for a list of commands.\n");
    while (true)
    {
        string str = Console.ReadLine().Trim().ToLower();
        if (str.Length == 0)
        {
            continue;
        }
```

```
        // initial blank space
        Console.WriteLine();

        if (str.CompareTo("help") == 0)
        {
            // lists all commands and their descriptions
            Console.WriteLine("Command\t\tDescription");
            Console.WriteLine("-------\t\t-----------");
            Console.WriteLine("players\t\tlists which players are
          currently connected");
            Console.WriteLine("help\t\tprovides a list of commands
          for using the server");
            Console.WriteLine("exit\t\tshuts down connections and
          closes the server software");
        }
        else if (str.CompareTo("exit") == 0)
        {
            // shuts down the server program
            broadcastToPlayers(SERVER_SHUTDOWN);
            Environment.Exit(0);
        }
        else if (str.CompareTo("players") == 0)
        {
            // display total current player connections
            Console.WriteLine("Players connected: " + numPlayers);

            // lists the player connections
            for (int i = 0; i < MAX_PLAYERS; i++)
            {
                Console.Write("Player " + i + ": ");
                if (players[i].active)
                {
                    Console.WriteLine("connected");
                }
                else
                {
                    Console.WriteLine("disconnected");
                }
            }
        }

        // ending blank space
        Console.WriteLine();
    }
}

// returns the number of PNG graphics present in graphics folder
int getNumGraphics()
{
    string filepath =

          Path.GetDirectoryName(Assembly.GetExecutingAssembly().GetNa
          me().CodeBase)
        .Substring(6) + "\\game_files\\graphics\\";
    DirectoryInfo dir = new DirectoryInfo(filepath);
    return dir.GetFiles("*.png").Length;
}
```

```
// returns the IPv4 address of the server
string getIP4Address()
{
    // Adapted from:
          http://aspnet.4guysfromrolla.com/articles/071807-1.aspx

    string IP4Address = String.Empty;

    foreach (IPAddress ip in Dns.GetHostAddresses(Dns.GetHostName()))
    {
        if (ip.AddressFamily.ToString() == "InterNetwork")
        {
            IP4Address = ip.ToString();
            break;
        }
    }

    if (IP4Address != String.Empty)
    {
        return IP4Address;
    }

    foreach (IPAddress IPA in
          Dns.GetHostAddresses(Dns.GetHostName()))
    {
        if (IPA.AddressFamily.ToString() == "InterNetwork")
        {
            IP4Address = IPA.ToString();
            break;
        }
    }

    return IP4Address;
}
#endregion

#region HYDRA methods
// connects to the HYDRA via serial-over-USB
unsafe void connectToHydra()
{
    Console.WriteLine();
    Console.WriteLine("Connecting to the HYDRA...");

    while (true)
    {
        // auto scan for port number
        int port = HydraPropForm.getHydraComPort();

        // create serial port
        hydra = new SerialPort("COM" + port, 9600, Parity.None, 8,
          StopBits.One);

        // open serial port
        try
        {
            hydra.Open();
```

```
                    break;
                }
                catch (Exception)
                {
                    while (true)
                    {
                        // check if user wants to try again
                        Console.Write("Could not connect to HYDRA. Try again?
                  [y/n]: ");
                        string s = Console.ReadLine().ToLower();
                        if (s.CompareTo("y") == 0)
                        {
                            break;
                        }
                        else if (s.CompareTo("n") == 0)
                        {
                            Console.WriteLine("Cannot connect to the HYDRA.
                  Closing server.");
                            Environment.Exit(0);
                        }
                    }
                }
            }

            Console.WriteLine("The server is connected to the HYDRA.");
            Console.WriteLine();
        }
        #endregion

        static void Main(string[] args)
        {
            new HydraServer();
        }
    }
}
```

Appendix B: HydraProp DLL Source Code (C#)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.IO;

namespace HydraProp
{
    public partial class HydraPropForm : Form
    {
        // prototypes to permit use of the Propellent.dll functions
        #region Propellent DLL prototypes
        [DllImport("Propellent.dll", CharSet = CharSet.Ansi,
                  CallingConvention = CallingConvention.Cdecl)]
        static extern unsafe void InitPropellent(IntPtr WinHandle, bool
                  StorePrefs, IntPtr RegPath);

        [DllImport("Propellent.dll", CharSet = CharSet.Ansi,
                  CallingConvention = CallingConvention.Cdecl)]
        static extern unsafe uint GetPropellerVersion();

        [DllImport("Propellent.dll", CharSet = CharSet.Ansi,
                  CallingConvention = CallingConvention.Cdecl)]
        static extern unsafe void SetSerialSearchMethod(uint Value);

        [DllImport("Propellent.dll", CharSet = CharSet.Ansi,
                  CallingConvention = CallingConvention.Cdecl)]
        static extern unsafe void SetResetSignal(byte Value);

        [DllImport("Propellent.dll", CharSet = CharSet.Ansi,
                  CallingConvention = CallingConvention.Cdecl)]
        static extern unsafe void DownloadToPropeller(string Filename, byte
                  DownloadCmd);

        [DllImport("Propellent.dll", CharSet = CharSet.Ansi,
                  CallingConvention = CallingConvention.Cdecl)]
        static extern unsafe void FinalizePropellent();
        #endregion

        // displays the window
        public HydraPropForm()
        {
            InitializeComponent();
            this.Activate();
            this.Show();
        }
```

```
// initializes the Propellent DLL file on startup
private void HydraPropForm_Load(object sender, EventArgs e)
{
    try
    {
        // initializes Propellent.dll
        InitPropellent(this.Handle, false, IntPtr.Zero);
    }
    catch (Exception)
    {
        // the Propellent.dll file does not exist so show error
          message and exit
        MessageBox.Show("The Propellent.dll file was not found.",
          "Error",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
        System.Environment.Exit(1);
    }
}

// finalizes the Propellent DLL file on shut down
private void HydraPropForm_FormClosing(object sender,
          FormClosingEventArgs e)
{
    FinalizePropellent();
}

// gets the COM port that the HYDRA is connected to or -1 if it is
          not found
public static int getHydraComPort()
{
    // create and display the window
    HydraPropForm hpf = new HydraPropForm();

    // auto scan for port number
    uint ver = HydraPropForm.GetPropellerVersion();

    // close the window
    hpf.Close();

    if (ver == 0)
    {
        // ver=0 when HYDRA was not found
        return -1;
    }
    else
    {
        // port number is in bits 31-16
        return (int)(ver >> 16);
    }
}

public static bool uploadGame(String filePath)
{
    // verify that the file exists and that it has the correct file
          extension
    FileInfo fi = new FileInfo(filePath);
    if (!fi.Exists && fi.Extension.ToLower() != "eeprom")
```

```
            {
                // upload could not continue
                return false;
            }

            // create and display the window
            HydraPropForm hpf = new HydraPropForm();

            // specifies auto search for serial port
            SetSerialSearchMethod(0);

            // specifies DTR for reset signal
            SetResetSignal(0);

            // upload the game file
            //string cfile = filePath.ToCharArray();
            DownloadToPropeller(filePath, 3);

            // close the window
            hpf.Close();

            // upload was successful
            return true;
        }
    }
}
```

Appendix C: Main Web Site (ASP.NET)

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
                  Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>HydraNet 3 Gaming System</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1>
            HydraNet 3 Gaming System</h1>
        <h3>
            CEN 4935 Senior Software Engineering Project (Spring 2009)<br />
            By Scott Marks</h3>
        <p>
            The purpose of this project is to expand and improve HydraNet 2
                  in two ways:
            the addition of game graphics and a web interface. Developers now
                  have the
            ability to create high-color game graphics for the HydraNet
                  system without a significant
            reduction in performance. The web interface, with the game
                  embedded as a Java
            applet, allows for a better user experience, both to players and
                  developers
            alike.</p>
        <p>
            View the entire <a href="docs/HydraNet3Report.doc">HydraNet 3
                  project report</a>
            entitled &quot;HYDRA Network Control and Graphics.&quot;</p>
        <h3>
            HydraNet 3 Player</h3>
        <p>
            The <a href="Player.aspx">Player page</a> can be used to play the
                  current game.</p>
        <p>
            View the <a href="Manual.aspx" target="_blank">game manual</a>
                  (opens in new
            window/tab).</p>
        <p>
            The game controls are:</p>
        <ul>
            <li>A: Left</li>
            <li>D: Right</li>
            <li>W: Up</li>
            <li>S: Down</li>
            <li>J: Action</li>
        </ul>
```

```
<h3>
    HydraNet 3 Developer</h3>
<p>
    The <a href="Developer.aspx">Developer page</a> can be used to
          create and upload
    your own games. This page also includes a tutorial on how to
          develop a game.</p>
<p>
     </p>
<h2>
    Past projects</h2>
<h3>
    HydraNet 2</h3>
<h4>
    CNT 4104 Computer Network Programming (Fall 2008)<br />
    By Scott Marks and Richard Haberkamp</h4>
<p>
    The purpose of this project was to develop a multiplayer network
          game using a
    single HYDRA. The players would connect to a PC server via player
          software on
    their computer. The players would send game inputs via the
          keyboard and the game
    screen would be displayed in the player software. The game
          followed the same
    format from HydraNet 1, but with two minor differences. First,
          the player
    sprites were simply numbers 1-4, indicating each player&#39;s
          number, instead of
    ghost-like figures. Second, the maximum number of concurrent
          players was
    increased from two to four.</p>
<p>
    View the entire <a href="docs/HydraNet2Report.doc">HydraNet 2
          project report</a>
    entitled &quot;HYDRA Network Control and Visualization.&quot;</p>
<h3>
    HydraNet 1</h3>
<h4>
    CEN 3213 Embedded Systems Programming (Spring 2008)<br />
    By Scott Marks and Richard Haberkamp</h4>
<p>
    The purpose of this project was to develop a two player HYDRA
          game that could be
    played using two HYDRAs connected over a network or the Internet.
          A computer
    program, which would act as either the server or client, handled
          the relaying of
    game state between HYDRAs. The game was very basic and included
          two sprites that
    moved around a black background. Each player had control over one
          sprite.</p>
<p>
    View the entire <a href="docs/HydraNet1Report.doc">HydraNet 1
          project report</a>
    entitled &quot;Programming an Internet Interface for the HYDRA
          Game Development Kit.&quot;</p>
```

```
        <p>
             </p>
    </div>
    </form>
</body>
</html>
```

Appendix D: Player Web Site (ASP.NET)

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Player.aspx.cs"
                  Inherits="Player" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>HydraNet 3 Player</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1>
            HydraNet 3 Player</h1>
        <p>
            Welcome to the HydraNet 3 Player. This page allows you to play
                  the currently
            uploaded game on the HYDRA.</p>
        <p>
            Go back to the <a href="Default.aspx">main page</a> or develop a
                  new game on the
            <a href="Developer.aspx">Developer page</a>.</p>
        <p>
            There is no need to worry about connecting and disconnecting. If
                  you are seeing
            this page and there was an open player slot on the server, then
                  you are already
            connected to the game. To disconnect, simply navigate away from
                  this page or
            close the browser window. To attempt to reconnect or to start
                  over, click the
            browser&amp;#39;s refresh button.</p>
        <p>
            View the <a href="Manual.aspx" target="_blank">game manual</a>
                  (opens in new
            window/tab).</p>
        <p>
            The game controls are:</p>
        <ul>
            <li>A: Left</li>
            <li>D: Right</li>
            <li>W: Up</li>
            <li>S: Down</li>
            <li>J: Action</li>
        </ul>
        <p>
            <i>To play, you must first click within the screen area.</i></p>
        <p style="border: solid 1px black; width: 240px;">
            <applet code="HydraPlayer.class" width="240px" height="180px">
                <PARAM NAME="cache_option" VALUE="no">
                <PARAM NAME="cache_archive" VALUE="">
```

```
            </applet></p>
        <p>
            If a developer uploads a new game while you are playing, simply
                  wait until you
            receive a message stating that the new game is running. Then,
                  refresh this web
            page to start the new game.</p>
        <p>
             </p>
    </div>
    </form>
</body>
</html>
```

Appendix E: Player Client Applet Source Code (Java)

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.Socket;
import java.net.UnknownHostException;

import javax.swing.*;

public class HydraPlayer extends JApplet implements KeyListener, Runnable,
                 ActionListener{
   // connection-related variables
   private Socket conn;  // connection with the server
   private InputStream connIn;  // incoming stream from server
   private OutputStream connOut;  // outgoing stream to server

   // graphics-related class and variables
   private final static int SCREEN_SIZE_X=240, SCREEN_SIZE_Y=180;
   class GraphicObj{
      Image img;
      boolean visible=false;
      int x=0, y=0;
   }
   private GraphicObj[] images;
   private int numImages;
   private Image graphicsBuffer;
   private String message="";

   // player input values
   private final static int INPUT_ON=240,   // player on/reset
                       INPUT_OFF=0,    // player off
                       INPUT_UP=128,   // up
                       INPUT_DOWN=64,  // down
                       INPUT_LEFT=32,  // left
                       INPUT_RIGHT=16, // right
                       INPUT_FIRE=48;  // fire

   // command message values
   private final static int PLAYER0_CONNECT = 241,
                       PLAYER1_CONNECT = 242,
                       PLAYER2_CONNECT = 243,
                       PLAYER3_CONNECT = 244,

                       PLAYER0_DISCONNECT = 245,
                       PLAYER1_DISCONNECT = 246,
                       PLAYER2_DISCONNECT = 247,
                       PLAYER3_DISCONNECT = 248,

                       DEV_UPLOADING = 251,
                       DEV_FINISHED = 252,

                       SERVER_SHUTDOWN = 254;
```

```
// this method is called first
public void init(){
   super.init();

   // create the graphics buffer
   graphicsBuffer=createImage(SCREEN_SIZE_X, SCREEN_SIZE_Y);

   try{
      // connect to server and get network I/O streams
      conn=new Socket(getDocumentBase().getHost(), 5309);
      connIn=conn.getInputStream();
      connOut=conn.getOutputStream();
   }catch(UnknownHostException ex){
      displayMessage("Cannot connect to server.", false);
      return;
   }catch(IOException ex){
      displayMessage("Cannot connect to server.", false);
      return;
   }

   try{
      // tell server that this is a player client
      connOut.write(1);
      connOut.flush();

      // check if the connection was accepted by the server
      if(connIn.read()==1){
         // server was not full, connection accepted

         // get the number of images and create the array
         if((numImages=connIn.read())==0){
            displayMessage("Error getting images.", false);
            return;
         }
         images=new GraphicObj[numImages];

         // this is used to wait for all images to load
         MediaTracker tracker=new MediaTracker(this);

         // create the images by downloading them from the server
         for(int i=0; i<numImages; i++){
            images[i]=new GraphicObj();
            images[i].img=getImage(getDocumentBase(),
                  "game_files/graphics/"+i+".png");
            tracker.addImage(images[i].img, 0);
         }

         // wait for all images to load
         try{
            tracker.waitForAll();
         }catch(InterruptedException e){
            displayMessage("Could not download graphics.", false);
            return;
         }

         // tell server that image download is complete
```

```
            connOut.write(1);
            connOut.flush();

            // notify the HYDRA that player is starting session
            connOut.write(INPUT_ON);
            connOut.flush();

            // receive player connect message
            readCommand(connIn.read());

            // start listening for key presses
            addKeyListener(this);

            // start receiving graphics info stream
            new Thread(this).start();
         }else{
            // server was full, connection denied
            displayMessage("Server is full. Try again later.", false);

            // close the connection
            stop();
         }
      }catch(IOException ex){

      }
   }

   // stores the next game screen into the graphics buffer
   public void updateGraphics(){
      Graphics g=graphicsBuffer.getGraphics();
      g.clearRect(0, 0, SCREEN_SIZE_X, SCREEN_SIZE_Y);

      if(images!=null){
         for(int i=0; i<numImages; i++){
            if(images[i].visible){
               // calculate image offset to its center
               int xOffset=images[i].img.getWidth(this)/2;
               int yOffset=images[i].img.getHeight(this)/2;

               g.drawImage(images[i].img, images[i].x-xOffset,
                     SCREEN_SIZE_Y-images[i].y-yOffset, this);
            }
         }
      }

      // display the message if it exists
      if(message.length()>0){
         g.setColor(Color.RED);
         g.drawString(message, 5, 15);
      }
   }

   // required for JApplet; paints the applet window
   public void paint(Graphics g){
      updateGraphics();
      g.drawImage(graphicsBuffer, 0, 0, this);
   }
```

```
// required for JApplet; runs when the applet closes
public void stop(){
   super.stop();

   try{
      // removes images from cache, which avoids the problem when a new
      // game is loaded some of the old images are displayed
      for(int i=0; i<images.length; i++){
         images[i].img.flush();
      }

      // notify HYDRA that player is ending session
      connOut.write(INPUT_OFF);
      connOut.flush();

      connIn.close();
      connOut.close();
      conn.close();
   }catch(IOException ex){
      return;
   }catch(NullPointerException ex){
      return;
   }
}

// receive graphics stream from HYDRA
public void run(){
   int rec=0, id=0;
   boolean visible=false;
   do{
      try{
         if((rec=connIn.read())<0){
            return;  // stream was terminated and -1 was returned
         }
      }catch(IOException ex){
         return;
      }
   }while(rec!=255);
   while(true){
      try{
         // get object's ID and visibility
         if((rec=connIn.read())<0){
            return;  // stream was terminated and -1 was returned
         }else if(rec==255){
            continue;  // end byte was received, start over on next packet
         }else if(rec>=240 && rec<=254){
            readCommand(rec);  // read a command byte
            continue;
         }else{
            // determine visibility
            if(rec >> 7==1){
               visible=true;
            }else{
               visible=false;
            }
```

```
            // save the id
            id=rec & 127;  // id & 0111_1111

            // save the visibility
            try{
               images[id].visible=visible;
            }catch(ArrayIndexOutOfBoundsException ex){
               continue;
            }
         }

         // get object's x-coordinate
         if((rec=connIn.read())<0){
            return;  // stream was terminated and -1 was returned
         }else if(rec==255){
            continue;  // end byte was received, start over on next packet
         }else if(rec>=241 && rec<=254){
            readCommand(rec);  // read a command byte
            continue;
         }else{
            images[id].x=rec;
         }

         // get object's y-coordinate
         if((rec=connIn.read())<0){
            return;  // stream was terminated and -1 was returned
         }else if(rec==255){
            continue;  // end byte was received, start over on next packet
         }else if(rec>=241 && rec<=254){
            readCommand(rec);  // read a command byte
            continue;
         }else{
            images[id].y=rec;
         }

         // read the end byte
         if((rec=connIn.read())<0){
            return;  // stream was terminated and -1 was returned
         }else if(rec>=241 && rec<=254){
            readCommand(rec);  // read a command byte
            continue;
         }
      }catch(IOException ex){
         return;
      }

      // redraw the game screen
      repaint();
   }
}

// reacts to commands received from the server
public void readCommand(int command){
   switch(command){
   case PLAYER0_CONNECT:
      displayMessage("Player 1 connected.", true);
      break;
```

```
   case PLAYER1_CONNECT:
      displayMessage("Player 2 connected.", true);
      break;
   case PLAYER2_CONNECT:
      displayMessage("Player 3 connected.", true);
      break;
   case PLAYER3_CONNECT:
      displayMessage("Player 4 connected.", true);
      break;
   case PLAYER0_DISCONNECT:
      displayMessage("Player 1 disconnected.", true);
      break;
   case PLAYER1_DISCONNECT:
      displayMessage("Player 2 disconnected.", true);
      break;
   case PLAYER2_DISCONNECT:
      displayMessage("Player 3 disconnected.", true);
      break;
   case PLAYER3_DISCONNECT:
      displayMessage("Player 4 disconnected.", true);
      break;
   case DEV_UPLOADING:
      displayMessage("Game stopped by developer.", false);
      numImages=0;  // set number of images to 0 to clear game screen
      break;
   case DEV_FINISHED:
      displayMessage("Refresh the page to start new game.", false);
      stop();  // disconnect from server
      break;
   case SERVER_SHUTDOWN:
      displayMessage("The server was shut down normally.", false);
      stop();  // disconnect from server
      break;
   }
}

// required for KeyListener but not used
public void keyPressed(KeyEvent arg0){ }
public void keyReleased(KeyEvent arg0){ }

// required for KeyListener; sends game inputs to HYDRA
public void keyTyped(KeyEvent ke){
   try{
      switch(Character.toUpperCase(ke.getKeyChar())){
         case 'A':
            connOut.write(INPUT_LEFT);
            break;
         case 'D':
            connOut.write(INPUT_RIGHT);
            break;
         case 'W':
            connOut.write(INPUT_UP);
            break;
         case 'S':
            connOut.write(INPUT_DOWN);
            break;
         case 'J':
```

```
                connOut.write(INPUT_FIRE);
                break;
             default:
                break;
          }
       }catch(IOException ex){
          return;
       }
    }

    // displays a message in the top left corner of the game screen; if timed
                    is
    // true, then the message will be displayed for only a short time
    public void displayMessage(String message, boolean timed){
       this.message=message;
       repaint();
       if(timed){
          Timer t=new Timer(3000, this);
          t.setRepeats(false);
          t.start();
       }
    }

    // required for ActionListener; clears the displayed message after a
                   certain time
    public void actionPerformed(ActionEvent e){
       message="";
       repaint();
    }
}
```

Appendix F: Developer Web Site (ASP.NET)

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Developer.aspx.cs"
                  Inherits="Developer"
    Culture="auto" meta:resourcekey="PageResource1" UICulture="auto" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>HydraNet 3 Developer</title>
    <style type="text/css">
        .style1
        {
            text-decoration: underline;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1>
            HydraNet 3 Developer</h1>
        <p>
            Welcome to the HydraNet 3 Developer. This page includes a
                  tutorial on how to develop
            games for HydraNet 3 and the means to upload your games to the
                  server.</p>
        <p>
            Go back to the <a href="Default.aspx">main page</a> or play the
                  current game on
            the <a href="Player.aspx">Player page</a>.</p>
        <h2>
            Game uploader</h2>
        <p>
            To upload a game for the HydraNet 3 system, three files are
                  required:</p>
        <ul>
            <li><span class="style1">Compiled game file</span>. This file
                  must be created using
                the Propeller Tool software, which is explained in the
                  tutorial. The file type must
                be EEPROM.</li>
            <li><span class="style1">Zipped images file</span>. This zip file
                  must contain only PNG images, which meet the requirements
                  in the tutorial. These images must be
                in the root directory of the zip file and the zip file must
                  not contain any subfolders.
                The file type must be ZIP.</li>
            <li><span class="style1">Game manual file</span>. This text file
                  can contain anything
                you want the players to know about the game. A link to this
                  document is provided
```

```
        on the main and player pages so that others can understand
          the rules and controls
        of the game. The file type must be TXT.</li>
</ul>
<p>
    Select the files and then click the &quot;Upload Files&quot;
          button:</p>
<p>
                Compiled game file (.eeprom)<br />
    <asp:FileUpload ID="upldGameFile" runat="server"
          meta:resourcekey="upldGameFileResource1" />
</p>
<p>
    Zipped images file (.zip)<br />
    <asp:FileUpload ID="upldImagesFile" runat="server"
          meta:resourcekey="upldImagesFileResource1" />
</p>
<p>
    Game manual file (.txt)<br />
    <asp:FileUpload ID="upldManualFile" runat="server"
          meta:resourcekey="upldManualFileResource1" />
</p>
<p>
    <asp:Button ID="btnUpload" runat="server"
          meta:resourcekey="btnUploadResource1"
          OnClick="btnUpload_Click"
        Text="Upload Files" CausesValidation="False" />
       <asp:Label ID="lblUploadStatus" runat="server"
          meta:resourcekey="lblUploadStatusResource1"
        Text="upload status" Visible="False" Style="font-style:
          italic; color: #CC0000"></asp:Label>
</p>
<p>
     </p>
<h2>
    Development tutorial</h2>
<h3>
    Download and install the Propeller IDE</h3>
<p>
    The files written in the Spin programming language for the HYDRA
          can be edited
    in a normal text editor such as Notepad or in the Propeller Tool.
          However, to upload the game to the HYDRA
    it must first be compiled using the Propeller Tool, which can be
          downloaded for
    free here:</p>
<dl>
    <dd>
        <a href="http://www.parallax.com/tabid/442/Default.aspx"
          target="_blank">Download the
            Parallax Propeller Tool Software and Propeller
          Manual</a></dd>
</dl>
<p>
    During the install, you will be given an option to also install a
          virtual COM port
```

```
    driver for use with the HYDRA console. This driver isn&#39;t
          necessary unless you
    have access to a HYDRA console.</p>
<p>
    The Propeller Manual can also be downloaded directly from that
          page, but it is normally
    included with the Propeller Tool software under the
          &quot;Help&quot; menu.</p>
<h3>
    Propeller&#39;s Spin language</h3>
<p>
    Games for the HYDRA, or more specifically Parallax&#39;s
          Propeller microcontroller,
    can be written in Spin, Propeller Assembly or both. Spin is an
          object-oriented language
    similar to C++ with some unique characteristics. The Propeller
          Tool software download
    includes a manual for programming the Propeller. This manual can
          be accessed from
    the Propeller Tool via the &quot;Help&quot; menu or directly from
          Parallax&#39;s
    Propeller download page (see link above).</p>
<h3>
    Source code template and two example games</h3>
<p>
    Two source files are required to operate a game on the HydraNet 3
          system:</p>
<ul>
    <li><a href="game_examples/game-template.spin">Game source
          template</a></li>
    <li><a href="game_examples/FullDuplexSerial.spin">HYDRA USB
          driver</a> (© Parallax,
        Inc. — also included with Propeller Tool download above)</li>
</ul>
<p>
    Two simple game examples are included for you to better
          understand how a game is built,
    or if you wish to restore the HydraNet 3 system to an original
          state:</p>
<ul>
    <li>Example 1 shows how to get 4 players to move about a static
          background.<ul>
        <li><a href="game_examples/game-ex1.eeprom">Compiled
          game</a></li>
        <li><a href="game_examples/game-ex1-code.spin">Source
          code</a></li>
        <li><a href="game_examples/game-ex1-images.zip">Images</a>
          (zipped)</li>
        <li><a href="game_examples/game-ex1-manual.txt">Manual</a><br
          />
        </li>
        </ul>
    </li>
    <li>Example 2 shows how to get 4 players to move between two
          backgrounds and how to
        get the player images to change based on movement and turbo
          state.<ul>
```

```
            <li><a href="game_examples/game-ex2.eeprom">Compiled
          game</a></li>
        <li><a href="game_examples/game-ex2-code.spin">Source
          code</a></li>
        <li><a href="game_examples/game-ex2-images.zip">Images</a>
          (zipped)</li>
        <li><a href="game_examples/game-ex2-
          manual.txt">Manual</a></li>
            </ul>
        </li>
</ul>
    <p>
        Both of these games require the HYDRA USB driver to compile,
          which is provided
        above with the game source template.</p>
<h3>
    Graphics</h3>
<p>
    Game graphics are implemented using object images stored on the
          server. The objects
    can then be turned on or off and moved around the game screen.
          Images are in PNG
    format, which was chosen primarily for its lossless compression
          and transparency.
    Overall, this provides easy creation of graphics with an image
          editor, coupled with
    the e editor, coupled with
    the <a href="http://www.libpng.org/pub/png/pngintro.html"
          target="_blank">advantages</a>
    of PNG.</p>
<p>
    Images must meet these requirements:</p>
<ul>
    <li>Images must be in PNG format</li>
    <li>Images must be numbered consectutively, starting at 0 and in
          the format &amp;quot;[number].png&amp;quot;
        (for example: &amp;quot;3.png&amp;quot;)</li>
    <li>A max of 100 images can be used (numbered 0-99)</li>
    <li>Higher numbered images display above lower numbered images on
          the game screen</li>
</ul>
<p>
    The game screen resolution is 240 x 180 pixels. Objects are
          positioned by their
    x- and y-coordinates, which indicate the position of the
          object&amp;#39;s image center
    on the game screen. Therefore, images can be placed on the game
          screen with as little
    as half the width and half the height visible. There are also no
          restrictions on
    each image size so if an object must be more than halfway
          offscreen, a transparent
    border can be placed around the image to make it look as though
          the object is almost
    (or completely) off the game screen. This permits development
          features such as scrolling
```

```
    backgrounds and wrap-around player movements.rounds and wrap-
          around player movements.</p>
<p>
    Players download all graphics from the server before starting the
          game so larger
    image file sizes, within reason, will not degrade the performance
          of the HydraNet
    3 game.</p>
<h3>
    Code sections for developers</h3>
<p>
    There are several sections in the template source code that are
          labeled for developer
    use:</p>
<ul>
    <li><span class="style1">Constant variables</span>. This section
          of code is for declaring
        any variables you need that will never change. Adding
          constant variables to this
        section is preferred over adding them to the regular
          variables section.</li>
    <li><span class="style1">Variables</span>. This section of code
          is for declaring any
        variables that may need to be updated during game
          execution.</li>
    <li><span class="style1">Start method</span>. This section of
          code is used for any initialization.
        The setGraphicsSize method must be called in this section.
          This method must be passed
        the total number of graphics object within the game. After
          calling this method,
        any initial graphics visibility or positioning should be
          handled.</li>
    <li><span class="style1">Update method</span>. This section of
          code is where the game
        occurs. Each time a player sends a game key press, this
          update method will be called
        to take the appropriate actions for the game to
          progress.</li>
    <li><span class="style1">Other methods</span>. This section is
          simply a placeholder
        for any additional methods you wish to include.</li>
</ul>
<p>
    Of course, the game requires much more code to operate, but these
          are the only sections
    that are needed for developing a new game. While the other code
          can be revised,
    deleted or added to, it is strongly discouraged since most of
          this code allows the
    HYDRA to communicate properly with the server and player
          components of the HydraNet
    3 system.</p>
<p>
    Please note that while the server and player applications show
          that the players
```

```
            are numbered 1-4, they are actually numbered 0-3 within the HYDRA
                  game template.
            For example, Player 1 on the server and player client corresponds
                  with Player 0
            in the game code.</p>
        <h3>
            Compile the game sources</h3>
        <p>
            Before uploading your game to the HydraNet 3 system, you need to
                  compile the game
            into an EEPROM file using the Propeller Tool software. With the
                  game source visible
            in the software, follow these steps:</p>
        <ol>
            <li>Click the &amp;quot;View Info...&amp;quot; option in the
                  &amp;quot;Run&amp;quot; -&amp;gt; &amp;quot;Compile
                Current&amp;quot; menu or press F8</li>
            <li>Click the &amp;quot;Show Hex&amp;quot; button</li>
            <li>Click the &amp;quot;Save EEPROM File&amp;quot; button</li>
            <li>Pick a file name and location</li>
            <li>Save the file</li>
        </ol>
        <h3>
            Game manual</h3>
        <p>
            This manual is meant to teach other players how to play your
                  game. There are no
            limitations to the game manual, other than it must be written as
                  a text file with
            the .txt extension. No HTML coding is required for text and
                  paragraph spacing; however,
            HTML tags for bold, italics, etc. can be used.</p>
        <p>
             </p>
    </div>
    </form>
</body>
</html>
```

Appendix G: Developer Client Source Code (C#)

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.IO;
using System.Net.Sockets;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using Ionic.Zip;

public partial class Developer : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        // do nothing
    }

    protected void btnUpload_Click(object sender, EventArgs e)
    {
        // verify that the files exist, else exit the method
        if (!upldGameFile.HasFile || !upldImagesFile.HasFile ||
                !upldManualFile.HasFile)
        {
            uploadStatus("ERROR: not enough files were uploaded.");
            return;
        }

        // verify that the files have the correct extensions, else exit the
                method
        if (!checkFileExt(upldGameFile.FileName, "eeprom") ||
            !checkFileExt(upldImagesFile.FileName, "zip") ||
            !checkFileExt(upldManualFile.FileName, "txt"))
        {
            uploadStatus("ERROR: at least one file does not have the correct
                  extension.");
            return;
        }

        // connect to the server
        TcpClient conn;
        Stream connStrm;
        try
        {
            conn = new TcpClient("localhost", 5309);
            connStrm = conn.GetStream();
        }
        catch (Exception)
```

```
    {
        uploadStatus("ERROR: could not connect to server.");
        return;
    }

    try
    {
        // tell server that this a developer client
        connStrm.WriteByte(0);

        // send notification that developer is ready to download
        connStrm.WriteByte(1);

        // receive notification that server has closed other connections
        if (connStrm.ReadByte() != 2)
        {
            throw new Exception();
        }

        // save uploaded files
        if (!uploadFiles())
        {
            // abort if upload failed; error message is already displayed

            // send notification that upload failed
            connStrm.WriteByte(0);

            // close connections and exit
            connStrm.Close();
            conn.Close();
            return;
        }

        // send notification that uploaded files are saved
        connStrm.WriteByte(3);

        // receive notification that game is uploaded to HYDRA
        if (connStrm.ReadByte() != 4)
        {
            throw new Exception();
        }
    }
    catch (Exception)
    {
        uploadStatus("ERROR: connection with server failed.");
        return;
    }
    finally
    {
        // close connection with server
        connStrm.Close();
        conn.Close();
    }

    // the upload was successful
    uploadStatus("Success!");
}
```

```
bool uploadFiles()
{
    // folder paths
    String rootDir = Page.Request.PhysicalApplicationPath,
        gameDir = rootDir + "game_files\\";

    // uploaded file paths
    String gameFile = gameDir + "game.eeprom",
        graphicsFile = gameDir + "graphics.zip",
        manualFile = gameDir + "manual.txt";

    // graphics folder path
    String graphicsFolder = gameDir + "graphics\\";

    // save the three files
    try
    {
        upldGameFile.SaveAs(gameFile);
        upldImagesFile.SaveAs(graphicsFile);
        upldManualFile.SaveAs(manualFile);
    }
    catch (Exception)
    {
        uploadStatus("ERROR: could not save uploaded files.");
        return false;
    }

    try
    {
        // delete the images from previous game
        if (Directory.Exists(graphicsFolder))
        {
            Directory.Delete(graphicsFolder, true);
        }

        // recreate the graphics directory
        Directory.CreateDirectory(graphicsFolder);

        // unzip the graphics file
        using (ZipFile zip = ZipFile.Read(graphicsFile))
        {
            foreach (ZipEntry zipEntry in zip)
            {
                // unzip each file in the zip file to the graphics
              folder,
                // overwriting when necessary
                zipEntry.Extract(graphicsFolder);
            }
        }
    }
    catch (Exception)
    {
        uploadStatus("ERROR: could not extract and save zipped images.");
        return false;
    }
```

```
        // upload was successful
        return true;
    }

    // displays the status of the upload
    void uploadStatus(string message)
    {
        lblUploadStatus.Text = message;
        if (message == "")
        {
            lblUploadStatus.Visible = false;
        }
        else
        {
            lblUploadStatus.Visible = true;
        }
    }

    // verifies that the file extension is correct
    bool checkFileExt(string fileName, string extension)
    {
        int dot = fileName.LastIndexOf(".") + 1;
        return fileName.Substring(dot).ToLower() == extension.ToLower();
    }
}
```

Appendix H: Game Manual Web Site (C#)

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Manual.aspx.cs"
                  Inherits="Manual" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>HydraNet 3 Game Manual</title>
    <style type="text/css">
        /* CSS code obtained from http://users.tkk.fi/~tkarvine/pre-wrap-
                  css3-mozilla-opera-ie.html */
        /* Browser specific (not valid) styles to make preformatted text wrap
                  */
        pre
        {
            white-space: pre-wrap; /* css-3 */
            white-space: -moz-pre-wrap; /* Mozilla, since 1999 */
            white-space: -pre-wrap; /* Opera 4-6 */
            white-space: -o-pre-wrap; /* Opera 7 */
            word-wrap: break-word; /* Internet Explorer 5.5+ */
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <pre><!--#include virtual="game_files\manual.txt" --></pre>
    </div>
    </form>
</body>
</html>
```

Appendix I: Hydra Game Example 1 Source Code (Spin)

```
' //////////////////////////////////////////////////////////////////////
' Game that allows 4 people to play from remote computers over a network
' //////////////////////////////////////////////////////////////////////

CON
  ' set clock mode/speed
  _clkmode = xtal1 + pll8x
  _xinfreq = 10_000_000

  ' screen size
  SCREEN_SIZE_X = 240
  SCREEN_SIZE_Y = 180

  ' max number of graphics objects allowed
  GRAPHICS_MAX = 100

  ' player input bit codes
  ' only uses first 4 bits since last 4 bits are for player flags
  INPUT_ON    = %1111_0000  ' player on/reset
  INPUT_OFF   = %0000_0000  ' player off
  INPUT_UP    = %1000_0000  ' up
  INPUT_DOWN  = %0100_0000  ' down
  INPUT_LEFT  = %0010_0000  ' left
  INPUT_RIGHT = %0001_0000  ' right
  INPUT_FIRE  = %0011_0000  ' fire

  ' max number of players allowed
  PLAYER_MAX = 4

  ' //////////////////////////////////////////////////////////////////
  ' Start section of code for developers: constant variables
  ' ----------------------------------------------------------------

  ' graphics object ID values
  IMG_BACKGROUND = 0
  IMG_PLAYER0 = 1
  IMG_PLAYER1 = 2
  IMG_PLAYER2 = 3
  IMG_PLAYER3 = 4

  ' ----------------------------------------------------------------
  ' End section of code for developers
  ' //////////////////////////////////////////////////////////////////

VAR
  ' number of graphics objects within game
  byte graphicsSize
  byte graphicsVisible[GRAPHICS_MAX]
  byte graphicsX[GRAPHICS_MAX]
  byte graphicsY[GRAPHICS_MAX]
```

```
  ' byte containing bit flags for each active player
  byte pStatus

  ' stack for receiving player inputs
  long receiveInputStack[100]

  ' stack for sending graphics
  long sendGraphicsStack[100]

  ' //////////////////////////////////////////////////////////////////
  ' Start section of code for developers: variables
  ' ----------------------------------------------------------------

  ' turbo for player movement
  byte playerTurbo[PLAYER_MAX]

  ' ----------------------------------------------------------------
  ' End section of code for developers
  ' //////////////////////////////////////////////////////////////////

OBJ
  usb: "FullDuplexSerial.spin"  ' USB driver

PUB start

  ' //////////////////////////////////////////////////////////////////
  ' Start section of code for developers: start method
  ' ----------------------------------------------------------------

  ' set the number of graphics objects
  setGraphicsSize(5)

  ' position the background
  setGraphicsObj(IMG_BACKGROUND, 1, getScreenCenterX, getScreenCenterY)

  ' ----------------------------------------------------------------
  ' End section of code for developers
  ' //////////////////////////////////////////////////////////////////

  ' start the game
  startGame

PUB update(player, input) | playerImg

  ' //////////////////////////////////////////////////////////////////
  ' Start section of code for developers: update method
  ' ----------------------------------------------------------------

  playerImg:=getPlayerGraphicObj(player)

  if input==INPUT_ON
    ' set the player as active
    setPlayerStatus(player, 1)
    setGraphicsObj(playerImg, 1, getScreenCenterX, getScreenCenterY)
    playerTurbo[player]:=0
  elseif input==INPUT_OFF
    ' set the player as inactive
```

```
    setPlayerStatus(player, 0)
    setGraphicsObjVisible(playerImg, 0)
  elseif input==INPUT_FIRE
    ' toggle the turbo
    if(playerTurbo[player]==0)
      playerTurbo[player]:=2
    else
      playerTurbo[player]:=0
  elseif input==INPUT_UP
    ' move player up
    setGraphicsObjY(playerImg,
                getGraphicsObjY(playerImg)+(2+playerTurbo[player]) <#
                getScreenSizeY)
  elseif input==INPUT_DOWN
    ' move player down
    setGraphicsObjY(playerImg, getGraphicsObjY(playerImg)-
                (2+playerTurbo[player]) #> 0)
  elseif input==INPUT_LEFT
    ' move player left
    setGraphicsObjX(playerImg, getGraphicsObjX(playerImg)-
                (2+playerTurbo[player]) #> 0)
  elseif input==INPUT_RIGHT
    ' move player right
    setGraphicsObjX(playerImg,
                getGraphicsObjX(playerImg)+(2+playerTurbo[player]) <#
                getScreenSizeX)

  ' ------------------------------------------------------------------
  ' End section of code for developers
  ' ////////////////////////////////////////////////////////////////


' ////////////////////////////////////////////////////////////////
' Start section of code for developers: developer-defined methods
' ------------------------------------------------------------------

PUB getPlayerGraphicObj(player)
  if player==0
    return IMG_PLAYER0
  elseif player==1
    return IMG_PLAYER1
  elseif player==2
    return IMG_PLAYER2
  elseif player==3
    return IMG_PLAYER3

' ------------------------------------------------------------------
' End section of code for developers
' ////////////////////////////////////////////////////////////////

PUB startGame

  ' start the USB connection
  usb.start(31, 30, 0, 9600) ' receive pin, transmit pin, mode, baud rate

  ' start a new cog to read all player inputs
  cognew(receiveInput, @receiveInputStack)
```

```
  ' start a new cog to send game state to players
  cognew(sendGraphics, @sendGraphicsStack)

PUB receiveInput | rec, player, input

  repeat
    ' receive the byte containing player number and input
    rec:=usb.rxtime(5)

    ' get player number (last 4 bits)
    player:=rec & %0000_1111

    ' get input code (first 4 bits)
    input:=rec & %1111_0000

    ' update the player's position
    update(player, input)

PUB sendGraphics | id, tempID
  ' send info on active graphics objects
  ' send id, x, y of each active graphics object

  repeat
    if pStatus>0  ' only send if players are connected
      repeat id from 0 to graphicsSize-1
        if getGraphicsObjVisible(id)
          tempID:=id | %1000_0000 ' indicates that the object is visible
        else
          tempID:=id

        ' send graphics ID number and visibility
        usb.tx(tempID)

        ' object's x-coordinate
        usb.tx(graphicsX[id])

        ' object's y-coordinate
        usb.tx(graphicsY[id])

        ' send end byte indicating that the next byte
        ' will belong to the next object
        usb.tx(%1111_1111)

PUB getGraphicsSize
  return graphicsSize

PUB setGraphicsSize(size)
  graphicsSize:=size

PUB setGraphicsObj(id, visible, x, y)
  setGraphicsObjVisible(id, visible)
  setGraphicsObjX(id, x)
  setGraphicsObjY(id, y)

PUB getGraphicsObjVisible(id)
  return graphicsVisible[id]
```

```
PUB setGraphicsObjVisible(id, visible)
  graphicsVisible[id]:=visible

PUB getGraphicsObjX(id)
  return graphicsX[id]

PUB setGraphicsObjX(id, x)
  graphicsX[id]:=x

PUB getGraphicsObjY(id)
  return graphicsY[id]

PUB setGraphicsObjY(id, y)
  graphicsY[id]:=y

PUB getPlayerMax
  return PLAYER_MAX

PUB getPlayerStatus(id) | temp
  ' checks to see if the player is active
  temp:=(1 << id) & pStatus
  return (temp >> id)

PUB setPlayerStatus(id, active)
  if active==1
    ' sets the player to be active
    pStatus:=(%0000_0001 << id) | pStatus
  else
    ' set the player to be inactive
    if id==0
      pStatus:= %1111_1110 & pStatus
    elseif id==1
      pStatus:= %1111_1101 & pStatus
    elseif id==2
      pStatus:= %1111_1011 & pStatus
    elseif id==3
      pStatus:= %1111_0111 & pStatus

PUB getScreenSizeX
  return SCREEN_SIZE_X

PUB getScreenSizeY
  return SCREEN_SIZE_Y

PUB getScreenCenterX
  return getScreenSizeX/2

PUB getScreenCenterY
  return getScreenSizeY/2
```

Appendix J: Hydra Game Example 2 Source Code (Spin)

```
' //////////////////////////////////////////////////////////////////////
' Game that allows 4 people to play from remote computers over a network
' //////////////////////////////////////////////////////////////////////

CON
  ' set clock mode/speed
  _clkmode = xtal1 + pll8x
  _xinfreq = 10_000_000

  ' screen size
  SCREEN_SIZE_X = 240
  SCREEN_SIZE_Y = 180

  ' max number of graphics objects allowed
  GRAPHICS_MAX = 100

  ' player input bit codes
  ' only uses first 4 bits since last 4 bits are for player flags
  INPUT_ON    = %1111_0000  ' player on/reset
  INPUT_OFF   = %0000_0000  ' player off
  INPUT_UP    = %1000_0000  ' up
  INPUT_DOWN  = %0100_0000  ' down
  INPUT_LEFT  = %0010_0000  ' left
  INPUT_RIGHT = %0001_0000  ' right
  INPUT_FIRE  = %0011_0000  ' fire

  ' max number of players allowed
  PLAYER_MAX = 4

  ' //////////////////////////////////////////////////////////////////
  ' Start section of code for developers: constant variables
  ' ----------------------------------------------------------------

  ' background images
  IMG_BG1 = 0
  IMG_BG2 = 1

  ' player 0 images
  IMG_P0L  = 2
  IMG_P0R  = 3
  IMG_P0LT = 4
  IMG_P0RT = 5

  ' player 1 images
  IMG_P1L  = 6
  IMG_P1R  = 7
  IMG_P1LT = 8
  IMG_P1RT = 9

  ' player 2 images
  IMG_P2L  = 10
```

```
  IMG_P2R  = 11
  IMG_P2LT = 12
  IMG_P2RT = 13

  ' player 3 images
  IMG_P3L  = 14
  IMG_P3R  = 15
  IMG_P3LT = 16
  IMG_P3RT = 17

  ' door area
  DOOR_X_MIN = 195
  DOOR_X_MAX = 225
  DOOR_Y_MIN = 55
  DOOR_Y_MAX = 95

  ' number of extra pixels to move when turbo is on
  TURBO_SIZE = 2

  ' ------------------------------------------------------------------
  ' End section of code for developers
  ' ////////////////////////////////////////////////////////////////////

VAR
  ' number of graphics objects within game
  byte graphicsSize
  byte graphicsVisible[GRAPHICS_MAX]
  byte graphicsX[GRAPHICS_MAX]
  byte graphicsY[GRAPHICS_MAX]

  ' byte containing bit flags for each active player
  byte pStatus

  ' stack for receiving player inputs
  long receiveInputStack[100]

  ' stack for sending graphics
  long sendGraphicsStack[100]

  ' ////////////////////////////////////////////////////////////////////
  ' Start section of code for developers: variables
  ' ------------------------------------------------------------------

  ' turbo toggle for player movement
  byte playerTurbo[PLAYER_MAX]

  ' stores current player image for each player
  byte playerImage[PLAYER_MAX]

  ' ------------------------------------------------------------------
  ' End section of code for developers
  ' ////////////////////////////////////////////////////////////////////

OBJ
  usb: "FullDuplexSerial.spin"  ' USB driver

PUB start
```

```
  ' ////////////////////////////////////////////////////////////////
  ' Start section of code for developers: start method
  ' ----------------------------------------------------------------

  ' set the number of graphics objects
  setGraphicsSize(18)

  ' position the backgrounds; 2nd background is invisible
  setGraphicsObj(IMG_BG1, 1, getScreenCenterX, getScreenCenterY)
  setGraphicsObj(IMG_BG2, 0, getScreenCenterX, getScreenCenterY)

  ' ----------------------------------------------------------------
  ' End section of code for developers
  ' ////////////////////////////////////////////////////////////////

  ' start the game
  startGame

PUB update(player, input) | pos

  ' ////////////////////////////////////////////////////////////////
  ' Start section of code for developers: update method
  ' ----------------------------------------------------------------

  if input==INPUT_ON
    ' set the player as active
    setPlayerStatus(player, 1)

    ' determine player's starting image (facing left with no turbo)
    if player==0
      playerImage[player]:=IMG_P0L
    elseif player==1
      playerImage[player]:=IMG_P1L
    elseif player==2
      playerImage[player]:=IMG_P2L
    elseif player==3
      playerImage[player]:=IMG_P3L

    ' enable player's image
    setGraphicsObj(playerImage[player], 1, getScreenCenterX,
                   getScreenCenterY)

    ' set turbo to off
    playerTurbo[player]:=0

  elseif input==INPUT_OFF
    ' set the player as inactive
    setPlayerStatus(player, 0)
    setGraphicsObjVisible(playerImage[player], 0)

  elseif input==INPUT_FIRE
    ' toggle the turbo
    if playerTurbo[player]==0
      setTurbo(player, 1)
    else
      setTurbo(player, 0)
```

```
  elseif input==INPUT_UP
    ' move player up
    pos:=getGraphicsObjY(playerImage[player])+(2+playerTurbo[player]) <#
                 getScreenSizeY-1
    setGraphicsObjY(playerImage[player], pos)
    checkDoor(getGraphicsObjX(playerImage[player]), pos)

  elseif input==INPUT_DOWN
    ' move player down
    pos:=getGraphicsObjY(playerImage[player])-(2+playerTurbo[player]) #> 0
    setGraphicsObjY(playerImage[player], pos)
    checkDoor(getGraphicsObjX(playerImage[player]), pos)

  elseif input==INPUT_LEFT
    ' move player left
    pos:=getGraphicsObjX(playerImage[player])-(2+playerTurbo[player]) #> 0
    setGraphicsObjX(playerImage[player], pos)
    setLeft(player)
    checkDoor(pos, getGraphicsObjY(playerImage[player]))

  elseif input==INPUT_RIGHT
    ' move player right
    pos:=getGraphicsObjX(playerImage[player])+(2+playerTurbo[player]) <#
                 getScreenSizeX-1
    setGraphicsObjX(playerImage[player], pos)
    setRight(player)
    checkDoor(pos, getGraphicsObjY(playerImage[player]))

  ' ------------------------------------------------------------------
  ' End section of code for developers
  ' ////////////////////////////////////////////////////////////////


' ////////////////////////////////////////////////////////////////
' Start section of code for developers: developer-defined methods
' ------------------------------------------------------------------

PUB checkDoor(x, y) | i
  if x>DOOR_X_MIN and x<DOOR_X_MAX and y>DOOR_Y_MIN and y<DOOR_Y_MAX
    ' toggle background 2
    if getGraphicsObjVisible(IMG_BG2)==0
      setGraphicsObjVisible(IMG_BG2, 1)
    else
      setGraphicsObjVisible(IMG_BG2, 0)

    ' set all players to center of screen and face them left
    repeat i from 0 to PLAYER_MAX-1
      if getPlayerStatus(i)==1
        setGraphicsObj(playerImage[i], 1, getScreenCenterX, getScreenCenterY)
        setLeft(i)

PUB setLeft(player) | turbo, newImage
  ' save the turbo status of the player
  turbo:=playerTurbo[player]/2

  ' determine the correct new image
```

```
  if player==0
    if turbo==1
      newImage:=IMG_P0LT
    else
      newImage:=IMG_P0L
  elseif player==1
    if turbo==1
      newImage:=IMG_P1LT
    else
      newImage:=IMG_P1L
  elseif player==2
    if turbo==1
      newImage:=IMG_P2LT
    else
      newImage:=IMG_P2L
  elseif player==3
    if turbo==1
      newImage:=IMG_P3LT
    else
      newImage:=IMG_P3L

  ' change the player's image to face left
  changePlayerImage(player, newImage)

PUB setRight(player) | turbo, newImage
  ' save the turbo status of the player
  turbo:=playerTurbo[player]/2

  ' determine the correct new image
  if player==0
    if turbo==1
      newImage:=IMG_P0RT
    else
      newImage:=IMG_P0R
  elseif player==1
    if turbo==1
      newImage:=IMG_P1RT
    else
      newImage:=IMG_P1R
  elseif player==2
    if turbo==1
      newImage:=IMG_P2RT
    else
      newImage:=IMG_P2R
  elseif player==3
    if turbo==1
      newImage:=IMG_P3RT
    else
      newImage:=IMG_P3R

  ' change the player's image to face right
  changePlayerImage(player, newImage)

PUB setTurbo(player, enable) | newImage
  ' do not update turbo if request would put turbo out of bounds
  if enable==1 & playerTurbo[player]==TURBO_SIZE | enable==0 &
                  playerTurbo[player]==0
```

```
    return

  ' set the turbo; will set 0 or TURBO_SIZE depending on enable equaling 0 or
                  1
  playerTurbo[player]:=enable*TURBO_SIZE

  if enable==1
    ' determine the player's correct turbo image
    newImage:=playerImage[player]+2
  else
    ' determine the player's correct regular image
    newImage:=playerImage[player]-2

  ' change the player's image
  changePlayerImage(player, newImage)

PUB changePlayerImage(player, newImage) | x, y
  ' save the (x,y) position of the player
  x:=getGraphicsObjX(playerImage[player])
  y:=getGraphicsObjY(playerImage[player])

  ' remove the current player's image
  setGraphicsObjVisible(playerImage[player], 0)

  ' enable the player's new image at (x,y)
  setGraphicsObj(newImage, 1, x, y)

  ' change player's stored image value
  playerImage[player]:=newImage

' ------------------------------------------------------------------
' End section of code for developers
' //////////////////////////////////////////////////////////////////

PUB startGame

  ' start the USB connection
  usb.start(31, 30, 0, 9600) ' receive pin, transmit pin, mode, baud rate

  ' start a new cog to read all player inputs
  cognew(receiveInput, @receiveInputStack)

  ' start a new cog to send game state to players
  cognew(sendGraphics, @sendGraphicsStack)

PUB receiveInput | rec, player, input

  repeat
    ' receive the byte containing player number and input
    rec:=usb.rxtime(5)

    ' get player number (last 4 bits)
    player:=rec & %0000_1111

    ' get input code (first 4 bits)
    input:=rec & %1111_0000
```

```
    ' update the player's position
    update(player, input)

PUB sendGraphics | id, tempID
  ' send info on active graphics objects
  ' send id, x, y of each active graphics object

  repeat
    if pStatus>0  ' only send if players are connected
      repeat id from 0 to graphicsSize-1
        if getGraphicsObjVisible(id)
          tempID:=id | %1000_0000 ' indicates that the object is visible
        else
          tempID:=id

        ' send graphics ID number and visibility
        usb.tx(tempID)

        ' object's x-coordinate
        usb.tx(graphicsX[id])

        ' object's y-coordinate
        usb.tx(graphicsY[id])

        ' send end byte indicating that the next byte
        ' will belong to the next object
        usb.tx(%1111_1111)

PUB getGraphicsSize
  return graphicsSize

PUB setGraphicsSize(size)
  graphicsSize:=size

PUB setGraphicsObj(id, visible, x, y)
  setGraphicsObjVisible(id, visible)
  setGraphicsObjX(id, x)
  setGraphicsObjY(id, y)

PUB getGraphicsObjVisible(id)
  return graphicsVisible[id]

PUB setGraphicsObjVisible(id, visible)
  graphicsVisible[id]:=visible

PUB getGraphicsObjX(id)
  return graphicsX[id]

PUB setGraphicsObjX(id, x)
  graphicsX[id]:=x

PUB getGraphicsObjY(id)
  return graphicsY[id]

PUB setGraphicsObjY(id, y)
  graphicsY[id]:=y
```

```
PUB getPlayerMax
  return PLAYER_MAX

PUB getPlayerStatus(id) | temp
  ' checks to see if the player is active
  temp:=(1 << id) & pStatus
  return (temp >> id)

PUB setPlayerStatus(id, active)
  if active==1
    ' sets the player to be active
    pStatus:=(%0000_0001 << id) | pStatus
  else
    ' set the player to be inactive
    if id==0
      pStatus:= %1111_1110 & pStatus
    elseif id==1
      pStatus:= %1111_1101 & pStatus
    elseif id==2
      pStatus:= %1111_1011 & pStatus
    elseif id==3
      pStatus:= %1111_0111 & pStatus

PUB getScreenSizeX
  return SCREEN_SIZE_X

PUB getScreenSizeY
  return SCREEN_SIZE_Y

PUB getScreenCenterX
  return getScreenSizeX/2

PUB getScreenCenterY
  return getScreenSizeY/2
```

Appendix K: Hydra Game Example 3 Source Code (Spin)

```
CON

  _clkmode = xtal1 + pll4x          ' enable external clock and pll
times 4
  _xinfreq = 40_000_000             ' set frequency to 10 MHZ
  _stack   = 40                     ' accomodate display memory and
stack

VAR

long  blink_stack[20]               ' allocate 20 longs for the task
stack

'////////////////////////////////////////////////////////////////////
//////////
PUB Start
' this is the first entry point the system will see when the PChip
starts,
' execution ALWAYS starts on the first PUB in the source code for
' the top level file

' spawn 2 COGs each with the Blink function and some stack space
COGNEW (Blink(2_000_000), @blink_stack[0])
COGNEW (Blink(20_000_000), @blink_stack[10])

'sit in infinite loop, that is do not release COG 0
repeat while TRUE

'////////////////////////////////////////////////////////////////////
//////////
PUB Blink(rate)
' this is the parallel function, it simple blinks the debug LED on the
' hydra, note is must set the direction output and then falls into
' an infinite loop and turns the LED on / off with a delay count.
' the interesting thing to realize is that the "rate" is sent as a
parm
' when we launch the COG, so there will be 2 COGs running this SAME
' infinite loop, but each with a differnet blink rate, the results
' will be a blinking light that has both one constant blink rate
' with another super-imposed on it

DIRA[0] := 1

repeat while TRUE
  OUTA[0] := !OUTA[0]
  waitcnt(CNT + rate)
```

Appendix L: Hydra Game Example 4 Source Code (Spin)

```
'
////////////////////////////////////////////////////////////////////////
///////
' EXPERIMENTAL MARS LANDER DEMO PROGRAM
' AUTHOR: Andre' LaMothe
' LAST MODIFIED: 5.10.06
' VERSION 1.1
' COMMENTS: implements simple lunar lander, collision works, landing
works
' fuel works, try adding sound and changing the terrain etc.
'
' CONTROLS: gamepad (must be plugged in)
' Start  = "Start"
' Thrust = "B"
' Rotate Right = "Dpad Right"
' Rotate Left  = "Dpad Left"

'
////////////////////////////////////////////////////////////////////////
///////


'
////////////////////////////////////////////////////////////////////////
///////
' CONSTANTS SECTION
'
////////////////////////////////////////////////////////////////////////
///////

CON

  _clkmode = xtal1 + pll8x              ' enable external clock and
pll times 4
  _xinfreq = 10_000_000 + 0000          ' set frequency to 10 MHZ plus
some error
  _stack = ($2400 + $2400 + $100) >> 2  'accomodate display memory and
stack

  ' graphics driver and screen constants
  PARAMCOUNT        = 14
  OFFSCREEN_BUFFER  = $3800           ' offscreen buffer
  ONSCREEN_BUFFER   = $5C00           ' onscreen buffer

  ' size of graphics tile map
  X_TILES           = 12
  Y_TILES           = 12
```

```
SCREEN_WIDTH      = 192
SCREEN_HEIGHT     = 192

' text position constants
HUD_X_POS         = -SCREEN_WIDTH/2 + 10
HUD_Y_POS         = SCREEN_HEIGHT/2 - 1*14

LANDERS_X_POS     = SCREEN_WIDTH/2 - 12/2*12
LANDERS_Y_POS     = SCREEN_HEIGHT/2 - 1*14

' lander physics model
MAX_LANDER_VEL             = 4
MAX_LANDER_TOUCHDOWN_YVEL  = 100
MAX_LANDER_TOUCHDOWN_XVEL  = 20

GRAVITY             = $0000_0100

' angular constants to make object declarations easier
ANG_0    = $0000
ANG_360  = $2000
ANG_240  = ($2000*2/3)
ANG_180  = ($2000/2)
ANG_120  = ($2000/3)
ANG_90   = ($2000/4)
ANG_60   = ($2000/6)
ANG_45   = ($2000/8)
ANG_30   = ($2000/12)
ANG_22_5 = ($2000/16)
ANG_15   = ($2000/24)
ANG_10   = ($2000/36)
ANG_5    = ($2000/72)

' constants for math functions
SIN      = 0
COS      = 1

' game states
GAME_STATE_INIT      = 0
GAME_STATE_MENU      = 1
GAME_STATE_START     = 2
GAME_STATE_RUN       = 3
GAME_STATE_CRASHED   = 4
GAME_STATE_LANDED    = 5
GAME_STATE_OVER      = 6

' game object states
OBJECT_STATE_DEAD    = $00_01
OBJECT_STATE_ALIVE   = $00_02
OBJECT_STATE_DYING   = $00_04
OBJECT_STATE_FROZEN  = $00_08

' control interface
```

```
  THRUST_BUTTON_ID = 1
  FIRE_BUTTON_ID   = 0

  ' control keys
  KB_LEFT_ARROW  = $C0
  KB_RIGHT_ARROW = $C1
  KB_UP_ARROW    = $C2
  KB_DOWN_ARROW  = $C3
  KB_ESC         = $CB
  KB_SPACE       = $20
  KB_ENTER       = $0D

  ' NES bit encodings
  NES_RIGHT  = %00000001
  NES_LEFT   = %00000010
  NES_DOWN   = %00000100
  NES_UP     = %00001000
  NES_START  = %00010000
  NES_SELECT = %00100000
  NES_B      = %01000000
  NES_A      = %10000000

  ' color constant's to make setting colors for parallax graphics
setup easier
  COL_Black       = %0000_0010
  COL_DarkGrey    = %0000_0011
  COL_Grey        = %0000_0100
  COL_LightGrey   = %0000_0101
  COL_BrightGrey  = %0000_0110
  COL_White       = %0000_0111

  COL_Blue       = %0000_1_101
  COL_Blue2      = %0001_1_101
  COL_Purple     = %0010_1_101
  COL_Magenta    = %0011_1_101
  COL_Magenta2   = %0100_1_101
  COL_Red        = %0101_1_101
  COL_Orange     = %0110_1_101
  COL_Brown      = %0111_1_101
  COL_Yellow     = %1000_1_101
  COL_YelGrn     = %1001_1_101
  COL_Green      = %1010_1_101
  COL_Green2     = %1011_1_101
  COL_Green3     = %1100_1_101
  COL_Cyan       = %1101_1_101
  COL_Cyan2      = %1110_1_101
  COL_Cyan3      = %1111_1_101


  ' each palette entry is a LONG arranged like so: color 3 | color 2 |
color 1 | color 0
  COLOR_0 = (COL_Black << 0)
```

```
  COLOR_1 = (COL_Green << 8)
  COLOR_2 = (COL_White << 16)
  COLOR_3 = (COL_Red   << 24)


'
////////////////////////////////////////////////////////////////////////
///////
' VARIABLES SECTION
'
////////////////////////////////////////////////////////////////////////
///////

VAR

  long  tv_status     '0/1/2 = off/visible/invisible           read-
only
  long  tv_enable     '0/? = off/on                            write-
only
  long  tv_pins       '%ppmmm = pins                           write-
only
  long  tv_mode       '%ccinp = chroma,interlace,ntsc/pal,swap write-
only
  long  tv_screen     'pointer to screen (words)               write-
only
  long  tv_colors     'pointer to colors (longs)               write-
only
  long  tv_hc         'horizontal cells                        write-
only
  long  tv_vc         'vertical cells                          write-
only
  long  tv_hx         'horizontal cell expansion               write-
only
  long  tv_vx         'vertical cell expansion                 write-
only
  long  tv_ho         'horizontal offset                       write-
only
  long  tv_vo         'vertical offset                         write-
only
  long  tv_broadcast  'broadcast frequency (Hz)               write-
only
  long  tv_auralcog   'aural fm cog                            write-
only

  word  screen[x_tiles * y_tiles] ' storage for screen tile map
  long  colors[64]                ' color look up table

  ' player's lander all in fixed point 16.16
  long lander_state
  long lander_x
  long lander_y
```

```
  long lander_wp_x      ' whole part
  long lander_wp_y

  long lander_dp_x      ' decimal part
  long lander_dp_y

  long lander_dx        ' velocity
  long lander_dy
  long lander_angle     ' angular direction

  long lander_fuel      ' fuel of lander
  long num_landers      ' number of landers left

  ' game state variables
  word game_state
  word game_counter1
  word game_counter2

  ' random stuff
  byte random_counter

  ' string/key stuff
  byte sbuffer[17]
  byte curr_key
  byte temp_key
  long data

  ' nes gamepad vars
  long nes_buttons

  ' terrain collision algorithm
  long pixel_data

'
////////////////////////////////////////////////////////////////////////
///////
' OBJECT DECLARATION SECTION
'
////////////////////////////////////////////////////////////////////////
///////
OBJ

  tv    : "tv_drv_010.spin"        ' instantiate a tv object
  gr    : "graphics_drv_010.spin"  ' instantiate a graphics object

'////////////////////////////////////////////////////////////////////////
//
' EXPORT PUBLICS
//////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////
//
```

```
PUB start | i, j, base, base2, dx, dy, x, y, x2, y2, last_cos,
last_sin

'
////////////////////////////////////////////////////////////////////////
///////
' GLOBAL INITIALIZATION
'
////////////////////////////////////////////////////////////////////////
///////

  'start tv
  longmove(@tv_status, @tvparams, paramcount)
  tv_screen := @screen
  tv_colors := @colors
  tv.start(@tv_status)

  'init colors, each tile has same 3 colors
  repeat i from 0 to 64
    colors[i] := COLOR_3 | COLOR_2 | COLOR_1 | COLOR_0

  'init tile screen
  repeat dx from 0 to tv_hc - 1
    repeat dy from 0 to tv_vc - 1
      screen[dy * tv_hc + dx] := onscreen_buffer >> 6 + dy + dx *
tv_vc + ((dy & $3F) << 10)

  ' reset game state
  game_state     := GAME_STATE_INIT

  'start and setup graphics
  gr.start
  gr.setup(12, 12, 96, 96, offscreen_buffer)

 ' BEGIN GAME LOOP
////////////////////////////////////////////////////////////

  repeat
   ' MASTER GAME STATE MACHINE
////////////////////////////////////////////////

    case game_state
      GAME_STATE_INIT: ' this state initializes all GAME related vas


        'initialize player's ship
        lander_state := OBJECT_STATE_ALIVE
        lander_x     := -20 << 16
        lander_y     := +60 << 16
        lander_dx    := 0
        lander_dy    := 0
        lander_angle := ANG_90
```

```
        lander_fuel  := 999
        num_landers  := 3

        ' seed random counter
        random_counter := 17

        ' initialize game state counters
        game_counter1  := 0
        game_counter2  := 0

        ' set initial state to menu
        game_state := GAME_STATE_MENU

      GAME_STATE_START: ' the game is ready to start, do any
housekeeping and run

        're-initialize player's ship
        lander_state := OBJECT_STATE_ALIVE
        lander_x     := -20 << 16
        lander_y     := +60 << 16
        lander_dx    := 0
        lander_dy    := 0
        lander_angle := ANG_90
        game_state := GAME_STATE_RUN

      GAME_STATE_MENU: ' the game is running...

        'clear bitmap
        gr.clear

        ' INPUT SECTION
////////////////////////////////////////////////////

        ' get nes controller buttons
        nes_buttons := NES_Read_Gamepad

        if (nes_buttons & NES_START)
          game_state := GAME_STATE_START

        ' END INPUT SECTION
////////////////////////////////////////////////////

        ' RENDERING SECTION
////////////////////////////////////////////////////

        'draw text
        gr.textmode(2,1,5,3)
        gr.colorwidth(2,0)

        ' draw fuel and num players
        gr.text(HUD_X_POS, HUD_Y_POS, @hud_string)
        gr.text(LANDERS_X_POS, LANDERS_Y_POS, @landers_string)
```

```
        itoa(lander_fuel, @sbuffer)
        gr.text(HUD_X_POS, HUD_Y_POS-12, @sbuffer)

       ' draw vector objects

       ' draw mountain, static for now
        gr.colorwidth(1,0)
        gr.plot(-96, -96)
        y := -80

        repeat i from 1 to 20
          ' update y
          y += mountain_scape[i-1]
          ' draw line
          gr.line(-96 + i*8,y)

        ' finally mountainscape back to horizon
        gr.line(96,-96)

        ' draw landing zone
        gr.colorwidth(3,0)
        gr.plot(-96 + landing_zone[0], -96 + landing_zone[2])
        gr.line(-96 + landing_zone[1], -96 + landing_zone[2])

       ' draw players ships left
       gr.colorwidth(3,0)
       repeat i from 0 to num_landers-1
         gr.vec(LANDERS_X_POS + i << 3 + 4, LANDERS_Y_POS - 4, $0012,
$2000 >> 2, @lander_model)

       ' draw start game text
       if (game_state == GAME_STATE_MENU and (++game_counter2 & $10))
         gr.textmode(2,1,6,5)
         gr.colorwidth(2,0)
         if (game_counter2 > 200)
           gr.text(4,0,@start_string)
         else
           gr.text(4,0,@title_string)

       'copy bitmap to display
       gr.copy(onscreen_buffer)

       ' END RENDERING SECTION
////////////////////////////////////////////////

      GAME_STATE_RUN: ' the game is running...

        'clear bitmap
        gr.clear

        ' INPUT SECTION
////////////////////////////////////////////////////
```

```
        ' get nes controller buttons
        nes_buttons := NES_Read_Gamepad

        ' get change in direction
        if (nes_buttons & NES_RIGHT)
          lander_angle -= ANG_5
        else
        if (nes_buttons & NES_LEFT)
          lander_angle += ANG_5

        ' bounds test ship angle
        if (lander_angle > ANG_360)
          lander_angle -= ANG_360
        elseif (lander_angle < 0)
          lander_angle +=ANG_360

        ' END INPUT SECTION
////////////////////////////////////////////////

        ' ANIMATION SECTION
////////////////////////////////////////////////

        ' apply thrust model if thruster is down
        if (lander_fuel > 0 and nes_buttons & NES_B)
          lander_dx += (last_cos := SinCos(COS, lander_angle) ~> 6 )
          lander_dy += (last_sin := SinCos(SIN, lander_angle) ~> 6 )
          ' expend fuel
          lander_fuel -= 1

        ' add gravity
        lander_dy -= GRAVITY

        ' scale down x,y, cache result for later computation
        x := lander_dx ~> 16
        y := lander_dy ~> 16

        ' test for maximum magnitude of thrust
        if ((x*x + y*y) > MAX_LANDER_VEL)
          ' slow ship down
          lander_dx -= (lander_dx ~> 7)
          lander_dy -= (lander_dy ~> 7)

        ' move player
        lander_x += (lander_dx)
        lander_y += (lander_dy)

        ' cache whole parts
        lander_wp_x := lander_x ~> 16
        lander_wp_y := lander_y ~> 16

        ' screen bounds test for player
```

```
        if (lander_wp_x > SCREEN_WIDTH/2)
          lander_x -= (SCREEN_WIDTH << 16)
        elseif (lander_wp_x < -SCREEN_WIDTH/2)
          lander_x += (SCREEN_WIDTH << 16)

        if (lander_wp_y > SCREEN_HEIGHT/2)
          lander_y -= lander_dy << 1
          lander_dy := 0
        elseif (lander_wp_y < -SCREEN_HEIGHT/2)
          lander_y += lander_dy << 1
          lander_dy := 0

        ' re-acquire velocity in new scale
        x := lander_dx ~> 10
        y := lander_dy ~> 10

        ' test for landing zone, the conditions are downward velocity
has to below a threshold, absolute value of
        ' horizontal velocity has to be below a threshold and the
angle of the ship must be within 5 degrees of
        ' straight up, don't want to break the landing gear, if a
landing is successful, a state change is noted here
        ' and processed down stream, also the coordinate structure of
the "landing zone" is used to determine if the ship
        ' is touching down on the landing zone, but we could have
tested for "green" which is what we paint the landing zone
        ' with, however, if we couldn't spare a color to "paint" the
landing zone for color collision detection then we would
        ' have to do it the hard way with geometry, so this is a good
exercise
        if ( (lander_wp_x > (landing_zone[0]-96)) and (lander_wp_x <
(landing_zone[1]-96)) and (lander_wp_y < (landing_zone[2]-92)) )
          if ( (y < 0) and (y*y < MAX_LANDER_TOUCHDOWN_YVEL) and (x*x
< MAX_LANDER_TOUCHDOWN_XVEL) )
            gr.colorwidth(1,0)
            gr.plot(0, 0)

            ' the eagle has landed
            game_state := GAME_STATE_LANDED

        ' END ANIMATION SECTION
////////////////////////////////////////////

        ' RENDERING SECTION
////////////////////////////////////////////////

        'draw text
        gr.textmode(2,1,5,3)
        gr.colorwidth(2,0)

        ' draw fuel and num players
        gr.text(HUD_X_POS, HUD_Y_POS, @hud_string)
```

```
        gr.text(LANDERS_X_POS, LANDERS_Y_POS, @landers_string)
        itoa(lander_fuel, @sbuffer)
        gr.text(HUD_X_POS, HUD_Y_POS-12, @sbuffer)

       ' draw vector objects

       ' draw mountain, static for now
        gr.colorwidth(1,0)
        gr.plot(-96, -96)
        y := -80

        repeat i from 1 to 20
          ' update y
          y += mountain_scape[i-1]
          ' draw line
          gr.line(-96 + i*8,y)

        ' finally mountainscape back to horizon
        gr.line(96,-96)

        ' draw landing zone
        gr.colorwidth(3,0)
        gr.plot(-96 + landing_zone[0], -96 + landing_zone[2])
        gr.line(-96 + landing_zone[1], -96 + landing_zone[2])

       ' ignore test if already landed
       if (game_state <> GAME_STATE_LANDED)
         ' test for player collision with terrain BEFORE we draw the
player and thruster which would cause erroneous results
         ' scan 4 pixels from centroid of ship down to pierce terrain
          repeat i from 0 to 3
            pixel_data := Get_Pixel2(96 + lander_wp_x, 96 -
lander_wp_y + i, offscreen_buffer)
            ' test the pixel data to see if there is a collision
terrain
            if (pixel_data == 1)
              ' collision has occured, set state to kill player
              game_state := GAME_STATE_CRASHED
              quit

       ' draw player last (on top of terrain)
       ' first draw thruster
       if ( (lander_fuel > 0) and (nes_buttons & NES_B) and (Rand &
$01))
         gr.colorwidth(2,0)
         gr.plot(lander_wp_x, lander_wp_y)
         gr.line(lander_wp_x - (last_cos ~> 8), lander_wp_y -
(last_sin ~> 8) )

       ' now ship
       gr.colorwidth(3,0)
```

```
       gr.vec(lander_wp_x, lander_wp_y, $0020, lander_angle,
@lander_model)

       ' draw players ships left
       gr.colorwidth(3,0)
       repeat i from 0 to num_landers-1
         gr.vec(LANDERS_X_POS + i << 3 + 4, LANDERS_Y_POS - 4, $0012,
$2000 >> 2, @lander_model)

       ' test if state changed to GAME_STATE_CRASHED
       if (game_state == GAME_STATE_CRASHED)
          ' print out crash
          gr.textmode(3,1,6,5)
          gr.colorwidth(2,0)

          ' used up a player
          if (num_landers == 1)
            gr.text(4,0,@over_string)
            game_state := GAME_STATE_OVER
          else
            num_landers--
            gr.text(4,0,@crash_string)

       ' check if landed
       if (game_state == GAME_STATE_LANDED)
          ' print out crash
          gr.textmode(3,1,6,5)
          gr.colorwidth(2,0)
          gr.text(4,0,@landed_string)

       'copy bitmap to display, we have to do this here to make sure
to see the last text rendered into the buffer
       gr.copy(onscreen_buffer)

       if (game_state == GAME_STATE_CRASHED)
          ' stall the event loop, normally you wouldn't want to do
this, but in this case its ok since
          ' we are trying to keep code short
          repeat 1_000_000
          ' transition back to starting state
          game_state := GAME_STATE_START
       else
         if (game_state == GAME_STATE_OVER)
            repeat 1_000_000
           ' transition back to initialization state
          game_state := GAME_STATE_INIT

       ' check if landed
       if (game_state == GAME_STATE_LANDED)
         repeat 1_000_000
         ' transition back to initialization state
         game_state := GAME_STATE_INIT
```

```
       ' END RENDERING SECTION
////////////////////////////////////////////


  ' END MAIN GAME LOOP REPEAT BLOCK
////////////////////////////////////////

'
//////////////////////////////////////////////////////////////////////
///////

PUB Plot_Pixel(x, y, video_buffer, color)               |
video_offset, pixel_value
  ' plot pixel calculation using BYTE aligned calcs 192x192 bitmap,
12x12 tiles
  video_offset := video_buffer + (x >> 4) * (192*4) + ((x & %1111) >>
2) + (y << 2)

 ' read pixel group from memory
  pixel_value := byte[video_offset]

  ' mask AND out target bits, so color mixing doesn't occur
  pixel_value := pixel_value & !(%00000011 << ((x & %11) << 1))

  ' OR color with pixel value
  pixel_value := pixel_value | (color << ((x & %11) << 1))

  ' write pixel back to memory
  byte[video_offset] := pixel_value

'
//////////////////////////////////////////////////////////////////////
///////

PUB Plot_Pixel2(x, y, video_buffer, color)              |
video_offset, pixel_value
  ' plot pixel calculation using LONG aligned calcs 192x192 bitmap,
12x12 tiles
  video_offset := (video_buffer >> 2) + (x >> 4) * (192) + y

  ' read pixel group from memory
  pixel_value := long[0][video_offset]

  ' mask AND out target bits, so color mixing doesn't occur
  pixel_value := pixel_value & !(%11 << ((x & %1111) << 1))

  ' OR color with pixel value
  pixel_value := pixel_value | (color << ((x & %1111) << 1))

  ' write pixel back to memory
```

```
  long[0][video_offset] := pixel_value


'
////////////////////////////////////////////////////////////////////////
///////

PUB Get_Pixel2(x, y, video_buffer)              | video_offset,
pixel_value
  ' get pixel calculation using LONG aligned calcs 192x192 bitmap,
12x12 tiles
  video_offset := (video_buffer >> 2) + (x >> 4) * (192) + y

  ' read pixel group from memory (16 pixels, 32-bits)
  pixel_value := long[0][video_offset]

  ' mask AND out target bits to extract bits in question
  pixel_value := pixel_value & (%11 << ((x & %1111) << 1))

  ' now shift bits back to right and return pixel value from screen
  pixel_value := pixel_value >> ((x & %1111) << 1)

  ' pixel value will be from 0..3
  return (pixel_value)


'
////////////////////////////////////////////////////////////////////////
///////

PUB Rand : retval
  random_counter := (1 + random_counter ^ random_counter << 6 ^
random_counter << 2 ^ random_counter << 7)
  retval := random_counter

'
////////////////////////////////////////////////////////////////////////
///////

PUB NES_Read_Gamepad : nes_bits        |  i

'
////////////////////////////////////////////////////////////////////////
///////
' NES Game Paddle Read
'
////////////////////////////////////////////////////////////////////////
///////
' reads both gamepads in parallel encodes 8-bits for each in format
' right game pad #1 [15..8] : left game pad #0 [7..0]
'
' set I/O ports to proper direction
```

```
' P3 = JOY_CLK      (4)
' P4 = JOY_SH/LDn   (5)
' P5 = JOY_DATAOUT0 (6)
' P6 = JOY_DATAOUT1 (7)
' NES Bit Encoding
'
' RIGHT  = %00000001
' LEFT   = %00000010
' DOWN   = %00000100
' UP     = %00001000
' START  = %00010000
' SELECT = %00100000
' B      = %01000000
' A      = %10000000

' step 1: set I/Os
DIRA [3] := 1 ' output
DIRA [4] := 1 ' output
DIRA [5] := 0 ' input
DIRA [6] := 0 ' input

' step 2: set clock and latch to 0
OUTA [3] := 0 ' JOY_CLK = 0
OUTA [4] := 0 ' JOY_SH/LDn = 0
'Delay(1)

' step 3: set latch to 1
OUTA [4] := 1 ' JOY_SH/LDn = 1
'Delay(1)

' step 4: set latch to 0
OUTA [4] := 0 ' JOY_SH/LDn = 0

' step 5: read first bit of each game pad

' data is now ready to shift out
' first bit is ready
nes_bits := 0

' left controller
nes_bits := INA[5] | (INA[6] << 8)

' step 7: read next 7 bits
repeat i from 0 to 6
 OUTA [3] := 1 ' JOY_CLK = 1
 'Delay(1)
 OUTA [3] := 0 ' JOY_CLK = 0
 nes_bits := (nes_bits << 1)
 nes_bits := nes_bits | INA[5] | (INA[6] << 8)

 'Delay(1)
' invert bits to make positive logic
```

```
nes_bits := (!nes_bits & $FFFF)

'
////////////////////////////////////////////////////////////////////////
///////
' End NES Game Paddle Read
'
////////////////////////////////////////////////////////////////////////
///////

'
////////////////////////////////////////////////////////////////////////
///////

PUB Delay (count)      | i, x, y, z
  ' delay count times inner loop length
  repeat  i from  0 to count

'
////////////////////////////////////////////////////////////////////////
///////

PUB SinCos(op, angle): xy
  if (op==COS)
    angle+=$800

  if angle & $1000
    if angle & $800
      -angle
    angle |= $7000
    xy := -word[angle << 1]
  else
    if angle & $800
      -angle
    angle |= $7000
    xy := word[angle << 1]

'
////////////////////////////////////////////////////////////////////////
///////
PUB itoa(value, string_ptr)     | base, factor, i
' converts an integer into an ASCIIZ, up to 99_999x
' string_ptr points to target storage to store converted string


' 10000's place
base := 10_000

repeat i from 5 to 2
  factor := value / base
  byte[0][string_ptr++] := hex_table [ factor ]
  value -= factor*base
```

```
  base /= 10

' 1's position
byte[0][string_ptr++] := hex_table [ value ]

' NULL terminate
byte[0][string_ptr++] := 0


'
////////////////////////////////////////////////////////////////////////
///////
' DATA SECTION
'
////////////////////////////////////////////////////////////////////////
///////

DAT

' TV PARAMETERS FOR DRIVER
////////////////////////////////////////////////

tvparams                long    0               'status
                        long    1               'enable
                        long    %011_0000       'pins
                        long    %0000           'mode
                        long    0               'screen
                        long    0               'colors
                        long    x_tiles         'hc
                        long    y_tiles         'vc
                        long    10              'hx timing stretch
                        long    1               'vx
                        long    0               'ho
                        long    0               'vo
                        long    55_250_000      'broadcast
                        long    0               'auralcog

' POLYGON OBJECTS
////////////////////////////////////////////////////////////

lander_model            word    $4000+ANG_0
                        word    50

                        word    $8000+ANG_120
                        word    50

                        word    $8000+ANG_180
                        word    10

                        word    $8000+ANG_240
                        word    50
```

```
                        word    $8000+ANG_0
                        word    50

                        word    0

' STRING STORAGE
////////////////////////////////////////////////////////

hex_table               byte    "0123456789ABCDEF"
hud_string              byte    "Fuel (lbs)",0          'text
landers_string          byte    "Landers",0             'text
start_string            byte    "Press Start",0         'text
title_string            byte    "Mars Lander",0         'text
crash_string            byte    "Crashed!",0            'text
over_string             byte    "Game Over",0           'text
landed_string           byte    "Landed",0              'text

' to show use of data driven rendering, the mars terrain and landing
zone are defined by data
' this way you can change them easily

' MOUNTAIN SCAPE delta Y positions for cheesy mountains! Notice, the
"0"s make the landing zone
mountain_scape          long 0, 10,3,7,-10,-2,15,20,10,-40, 0,0, 15,
10,3, -30, -10, 5, 6, 10,-20

' landing zone data structure in format, x1, x2, y, shows up as green
on top of mars surface dirt
landing_zone            long 10*8, 12*8, 30
```